GAMMON, AN APPROACH TO THE CONCEPT OF STRATEGY IN GAME-PLAYING PROGRAMS

By

William Edward Bushey

United States Naval Postgraduate School

THESIS

GAMMON, AN APPROACH TO THE CONCEPT OF STRATEGY IN GAME-PLAYING PROGRAMS

by

William Edward Bushey

December 1970

This document has been approved for public release and sale; its distribution is unlimited.

T137104

Gammon, an Approach to the Concept of Strategy in Game-Playing Programs

by

William Edward Bushey
Lieutenant Commander, United States Navy
B.A., Williams College, 1958

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1970

Thesis B924
c.1

LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIF. 93940

ABSTRACT

Gammon is a computer program which plays Backgammon. It was developed to investigate a concept of strategy as an integrating and driving force in the play of the game. A strategy is defined and implemented as a set of tactics where each tactic is a specific feature of play. Moves are generated and evaluated on the basis of the degree to which they accomplish the objectives of the tactics making up the current strategy. Strategies are selected and changed during the course of a game by heuristic analysis of the current game situation. Deductive learning mechanisms are employed to improve the program's performance, against both human and virtual machine opponents.

TABLE OF CONTENTS

I. INTRODUCTION ---------- 5

A. PURPOSE ---------- 5

B. GENERAL CHARACTERISTICS ---------- 5

C. SUMMARY OF RESULTS AND CONCLUSIONS ---------- 6

II. BACKGROUND ---------- 8

III. THE GAME OF BACKGAMMON ---------- 13

IV. DESCRIPTION OF THE PROGRAM ---------- 18

A. GENERAL ---------- 18

B. BOARD REPRESENTATION ---------- 19

C. GAME STATISTICS ---------- 21

D. THE TACTICS OF GAMMON ---------- 23

E. THE STRATEGIES OF GAMMON ---------- 24

F. MOVE GENERATION AND EVALUATION ---------- 26

G. RISK EVALUATION ---------- 28

H. STRATEGY SELECTION ---------- 29

I. THE PROGRAM MODULES ---------- 30

J. GENERAL PROGRAM FLOW AND LOGIC ---------- 32

K. LEARNING MECHANISMS ---------- 36

L. MODES OF PLAY AND PROGRAM OPTIONS ---------- 38

M. ENVIRONMENT ---------- 40

V. RESULTS AND CONCLUSIONS ---------- 41

A. PLAY OF STRATEGIES ---------- 41

B. STRATEGY SELECTION ---------- 41

C. LEARNING ---------- 42

D. MAJOR AREAS OF DIFFICULTY ---------- 43

E. MAJOR STRENGTHS AND WEAKNESSES ---------- 43

F. SUGGESTED IMPROVEMENTS ---------- 44

G. CONCLUSIONS ---------- 45

APPENDIX A. LIST OF GAMMON TACTICS ---------- 47

APPENDIX B. SAMPLE STRATEGY SET ---------- 49

APPENDIX C. DESCRIPTIONS OF THE PROGRAM MODULES ---------- 50

SAMPLE COMPUTER GAME ---------- 60

PROGRAM LISTING ---------- 86

LIST OF REFERENCES ---------- 138

INITIAL DISTRIBUTION LIST ---------- 139

FORM DD1473 ---------- 140

I. INTRODUCTION

A. PURPOSE

This thesis describes an investigation into the use of strategies in a game-playing computer program. The vehicle for the study was Gammon, a program which plays Backgammon.

The research was motivated by four related factors.

1. The study of game strategies had not been a major objective of previous game-playing programs.

2. Because Backgammon play is characterized by the existence of several traditional and well-defined styles or strategies, it was especially suited to the study.

3. Experiments in machine learning could focus not only on the play of a given individual strategy, but on the process of selecting a strategy as well.

4. To the author's knowledge Backgammon had not yet been programmed for play on a computer.

B. GENERAL CHARACTERISTICS

Gammon has these principal characteristics:

1. The concept of strategy is central to the organization and operation of the program. The program analyzes the game situation, chooses an appropriate strategy, and selects the move most consistent with the definition of that strategy.

2. The basic logic of the program is that of goal-fulfillment. Strategies are defined operationally as sets of tactics or game features. Each tactic has an objective, the achievement of which

is the sole purpose of the mechanism which generates and evaluates moves. Unlike many other game-playing programs, Gammon does not construct and search a game tree.

3. Gammon assesses the consequences of proposed moves by incorporating into its evaluation process an estimate of the probable net worth of men captured or threatened.

4. Gammon improves its performance through a learning mechanism whereby a human "tutor" evaluates and supplements the responses of the program during the course of play.

5. The program operates in either an interactive or batch environment, and requires approximately 100K bytes on an IBM 360/67 (TSS). It was written in PL/I.

C. SUMMARY OF RESULTS AND CONCLUSIONS

In general, the validity of the strategy concept was verified by the performance of the program.

The strategies employed by the program were designed to emulate the traditional Backgammon styles of play. The play of Gammon generally conformed to these traditional styles, but lacked consistency. The uneveness of play was particularly noticeable when the learning mechanism was employed intensively at one stage of a game, without being balanced at other stages.

Perhaps the most serious weakness observed was the frequent inability of the program to capitalize on potentially highly favorable situations or to avoid potentially highly dangerous ones. These critical situations were usually short-range in nature, that is, existent only over a span of a few moves.

Thus, the strategy concept developed for Gammon was generally satisfactory in terms of the long-range or overall character of its play, but generally inadequate with respect to recognizing and exploiting critical short-range objectives.

Suggested improvements include refining the existing mechanisms for generating and evaluating moves, defining new tactics, and providing the capability to identify, store, and retrieve generalized critical board situations for use as temporary objectives.

The processing time required by Gammon was reasonable. The average response time at the terminal for a move cycle was less than a minute. The CPU time required for a complete game in the batch mode was approximately one minute.

II. BACKGROUND

Artificial intelligence research is concerned with constructing computer programs which exhibit behavior that is called "intelligent behavior" when observed in human beings. Within this field, game playing has been both a favorite and prominent endeavor.

> Game playing has many fascinating aspects to the researcher. Effectively, it provides a direct contest between man's wit and machine wit. On a more serious level, game situations provide problem environments which are relatively highly regular and well defined, but which afford sufficient complexity in solution generation so that intelligence and symbolic reasoning skills play a crucial role. In short, game environments are very useful task environments for studying the nature and structure of complex problem-solving processes [Ref. 1].

To date, programs have been written for a variety of games including chess, checkers, Go, Nim, Kalah, and bridge. Common to many of these programs are certain principal features.

All programs have some kind of scheme for evaluating or measuring the "goodness" of the move, play, or board configuration under consideration. Perhaps the most widely used technique is that of the linear scoring polynomial, which has the form

$$S = C_1X_1 + C_2X_2 + \ldots + C_n X_n.$$

The X's usually represent the value of some feature of play such as control of certain key board positions, the capture of an enemy piece, or the presence of a certain geometrical configuration of pieces. The C's are weighting coefficients which usually indicate the desirability of that particular feature.

Another common device is the game tree. All two-person zero-sum games (of which chess, checkers, and backgammon are examples) can be conceived of in terms of a tree structure which represents all possible sequences of moves. The main problem is one of assigning values to the nodes of the tree and of searching for a "best" path. A common technique is to explore the tree to certain depth, use the scoring technique to assign values at these points and then "back up" the score. That is, work the scores of the highest valued moves up through the tree to the present position. The well known checkers program of Samuel uses this technique [Ref. 2].

The complete game tree is almost always much too large to be stored or manipulated in any present day computer. Thus the requirements of practicality have given rise to sets of powerful heuristics for "pruning" the tree, or in some way reducing the dimensions of the solution space to be searched. It is the nature of heuristics that they are not always successful, but the necessity for having them has resulted in some of the most inventive and brilliant programming.

Another basis of organizing a game-playing program is the use of goals, features of the game which it is desirable to attain during the course of play. The chess program of Newell, Shaw, and Simon uses this approach [Ref. 3]. Their program uses a set of goals which are conceptual units of chess, for example, King safety, material balance, and control of the center. For each goal there are three routines associated with it: (1) a move generator that finds moves positively related to carrying out the goal; (2) a procedure for making a static evaluation of any position with respect to the goal; (3) an analysis move generator that explores the game tree in order to determine the possible consequences of a move.

Many game-playing programs, representing as they do a well defined subsystem of thought, are developed to explore the capabilities of the machine to exhibit learning behavior. Generally, learning mechanisms can be grouped into three types: deductive, inductive, and discovery. In the deductive method the programmer fixes the weights--the coefficients of the scoring polynomial--and the characteristics to be measured--the X's. In the inductive approach the programmer fixes the characteristics, but the coefficients are generated and adjusted at run times. Both the characteristics and weights are generated at run time using the discovery method. The Newell-Shaw-Simon program is deductive, while Samuel's program is inductive. BOGART [Ref. 4], a more generalized game player and problem solver, uses the discovery method when playing Tic-tac-toe by analyzing and generalizing on geometrical configurations of the board.

A prominent feature of many games, but one that has been absent from virtually all game playing programs, is the concept of strategy.

Samuel, in the second report [Ref. 5] on his checkers program, states:

> The chief defect of the program in the recent past, according to several checker masters, seems to have been its failure to maintain any fixed strategy during play. The good player during his own play will note that a given board situation is favorable to him in some one respect and perhaps unfavorable in some second respect, and he will follow some fairly consistent policy for several moves in a row. In general he will try to maintain his advantage and at the same time to overcome the unfavorable aspect. In doing this he may more or less ignore other secondary properties which, under different circumstances, might themselves be dominant. The program, as described, treats each board situation as a new problem.

However, in the same report Samuel suggests an approach to strategy not unlike that developed for Gammon. Samuel recognized

that the linear polynomial method of scoring moves suffered the disadvantage of assumed linear relationships between measured characteristics and the desirability of a move. In an attempt to correct this weakness Samuel employed what he calls "signature tables." These tables, in fact, are a technique for measuring the contribution to a move score of the interaction among members of sets of features. According to Samuel:

> A possible mechanism for introducing this kind of strategic planning is provided by the signature table procedure and by the plausibility analysis. It is only necessary to view the different signature types as different strategic elements and to alter the relative weights assigned to the different signature types as a result of the plausibility analysis of the initial board situation. For this to be effective, some care must be given to the groupings of the parameters into the signature types so that these signature types tend to correspond to recognizable strategic concepts. Fortunately, the same initial-level grouping of parameters that is indicated by interdependency considerations seems to be reasonable in terms of strategies. We conclude that it is quite feasible to introduce the concept of strategy in this restricted way.

Samuel subsequently abandoned this approach, suggesting that the disappointing results were due to "... the ineffectual arrangement of terms into usable strategic groups...."

With this background in mind, the overall design of Gammon can be placed in a more clear perspective. First, the method of generating and evaluating moves in Gammon is clearly more similar to the approach of Newell, Shaw, and Simon, than to the game tree technique. However, Gammon uses a set of features, i.e., a strategy, as a goal, and employs move generators and evaluators at the component level. The risk evaluator of Gammon performs a similar function as the analysis generators of Newell, Shaw, and Simon in assessing the consequences of a proposed move.

The concept of a strategy as a logical grouping of game features, as suggested by Samuel, is developed more fully in Gammon. In the latter, however, the intent is not to develop an improved scoring technique, but to give operational definitions to the styles of play or strategies traditional to the game of Backgammon. In Chapter IV this concept will be described in more detail, but at this point a brief look at the game of Backgammon is in order.

III. THE GAME OF BACKGAMMON

Backgammon is one of the oldest of board games, having been known to the Greeks and Romans of ancient times and praised by Chaucer and Shakespears. It is still a popular pastime in the Mediterranean area. The game enjoyed a resurge of popularity in this country during the 1930's when certain variants of the game increased its attractiveness as a kind of parlor gambling game. A variant of the game, "Acey-Deucey", remains popular in the U. S. Navy.

Belonging to the class of "military" board games, along with chess and checkers, Backgammon is a game of retreat. Opposing forces are deployed in the field with the objective being to move them safely through the counter-retreating enemy to one's "home" and thence off the board.

The figure below depicts the board, the initial positions, and the direction of movement.

The game is played on a board divided into two halves, or tables, by a perpendicular strip called the bar. Extending from the sides nearest the players toward the center are alternately colored narrow triangles, or points, to and from which the men move. The points are usually referred to by number, each player's points numbered from 12 to 1 from his outer to inner table. Each player has 15 men, usually white or red for one player and black for the other. A pair of dice completes the necessary equipment.

A play is made in Backgammon by moving a man, or men, as many points as there are pips on the thrown dice. If doublets are

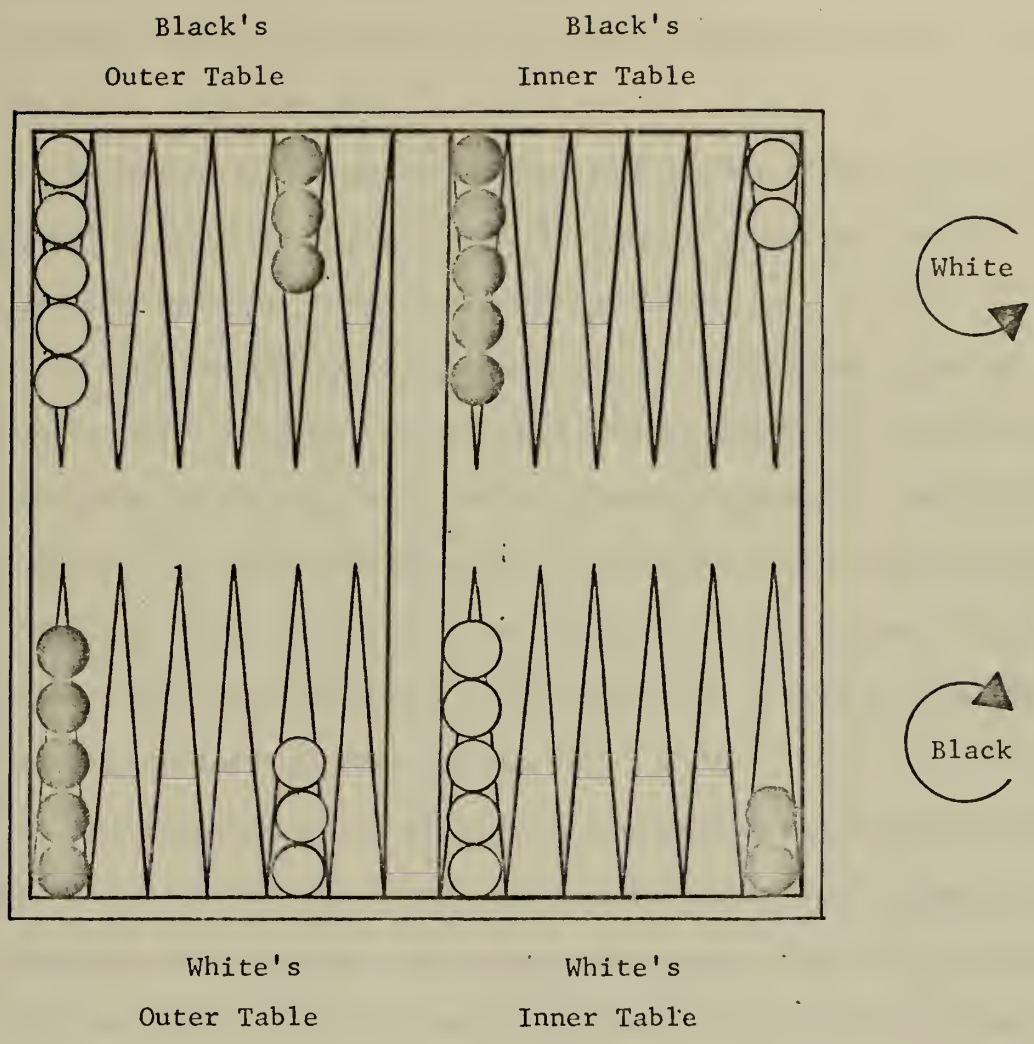

Figure 1. Backgammon Board

thrown, the player moves twice the number of points indicated. In playing a throw the number on each die is played separately, not the total as a single number.

In moving a man around the board a player cannot move a man to a point on which two or more of the opponent's men are resting. Such a point is called a "held" or "made" point.

If a player moves to a point occupied by only one opponent (a single man is termed a "blot") that man is called "hit" and is taken out of play by placing it on the bar. On the opponent's next play, before he can move any other men, all of his men on the bar must be brought back on the board by entering them in his opponent's inner table on the points corresponding to the dice roll. A man hit must thus start his journey all over again.

The objective of the game is to move all of one's men around the board and into his inner table while at the same time impeding the progress of his opponent as much as possible. Once all of a player's men are in his inner table he begins the final operation, known as "bearing off". This consists of removing men from the inner table and off the board according to the throws of the dice. The first player to do this wins the game.

A unique feature of Backgammon is the existence of several traditional strategies or styles of play.

1. The Position Game. This is perhaps the fundamental strategy of Backgammon. It is the cautious, conservative game in which one lets the opponent take the bold risks while you occupy key points, expose as few men as possible, bring up your rear-most men at every safe opportunity, and keep alert for advantageous

circumstances to switch to another strategy. Position is the strategy usually adopted for the early stages of the game.

2. The Running Game. This style of play is more aggressive, in which the object is to get as many men "home", i.e., into one's own inner table as expeditiously as possible. One accepts a higher risk in this game, although it can also be played defensively, with more concern for exposed men. This strategy is frequently adopted when one has an early advantage because of fortuitous dice rolls. It is also a common strategy for use near the end game when all or most of one's men have passed safely by the last opponent.

3. The Blocking Game. The object of this game is to erect a block or "side prime" of six consecutive points, thereby trapping the opponent's men in front. Ideally, the block is then advanced a point or two at a time, continuously impeding the progress of the opponent. The blocking strategy requires that the disposition of one's men make the construction of a block possible. Given these conditions the block is a powerful weapon, although it is more typically used when one is behind.

4. The Back Game. This is the strategy of desparation, to be used only when one is far behind in a game. In some respects it is a special case of the blocking game, in that the object is to build a block in or near the opponent's inner table, while continually hitting his blots whenever possible, and at the same time exposing any of your advanced men so that they may be hit and hence brought on the board near where the back block is being erected. It is a very difficult game to carry off successfully but is sometimes the only recourse.

5. Bearing Off. This is the end-game of Backgammon in which men are withdrawn from the board. Under no threat of capture the object is to get as many men off the board with as few rolls of the dice as possible. If, however, the opponent has some deep runners or men still on the bar, the bearing off process must be balanced against the precaution of not exposing men to capture. Hence there are distinct offensive and defensive end-game strategies.

References 6 through 8 contain complete descriptions of the rules and strategies of Backgammon.

IV. DESCRIPTION OF THE PROGRAM

A. GENERAL

As a computer program, Gammon has two significant characteristics: It is modular in design and operation, and the same set of modules can be used for either interactive or batch execution.

The modular approach was adopted in order to be able to call one module from within several other modules. Moreover, because of the large size of the program, the ability to design, write, compile, test, and execute it as logically separate entities was a practical as well as organizational necessity.

Each module is described in detail in Appendix C, but it is appropriate to list here the major functions performed by the program as a whole. These functions are:

1. To implement the various commands, options, and moves entered at the terminal.

2. To administer the game by calling the other modules in the appropriate sequence, and to gather and rearrange data for the arguments. This is the "main program" function.

3. To set up the initial board and initialize game statistics and variables.

4. To generate random numbers for use as the dice roll.

5. To select the strategy for the next move.

6. To provide a mechanism by which other modules can determine, for any roll of the dice, the points on the board to which, and from which, legal moves may be made.

7. To generate moves for the selected strategy and record them in a coded form.

8. To decode the moves and place them in a conventional list.

9. To evaluate the moves in the list and assign scores.

10. To locate the move with the highest score and rearrange the list for further evaluation if necessary.

11. To implement the selected move and update the game board and the cumulative statistics.

12. To print the resulting board.

13. To adjust coefficients after a move has been graded by the terminal user.

14. To perform a variety of utility functions such as displaying or searching the list of moves, and printing or storing the tactics and coefficients.

B. BOARD REPRESENTATION

Representing the Backgammon board internally is a relatively simple matter. Unlike many games which utilize a two-dimensional board, Backgammon pieces move in one dimension only. Hence, the board can be represented as a linear array of size 24, one element for each point on the board. Since only one player may occupy a given point the number of pieces on any point can be represented by an integer, positive for White pieces, negative for Black. The points are numbered from 1 to 24 beginning in White's inner table, and continuing through his outer table and Black's outer and inner tables. While this method of numbering is reversed in the usual mathematical

sense, it conforms to the customary notation of Backgammon literature.

In portraying the Backgammon board at the terminal or on printed output the following conventions were adopted:

1. The board is represented with the two halves joined in linear fashion with White's inner table and point 24 to the left. White moves from right to left, and Black from left to right.

2. White pieces are represented by an "O", Black pieces by an "X". To reduce the physical size of the board a maximum of two O's or X's are displayed on any one point. Any men in excess of two are denoted by an integer placed immediately above that point.

The figure below is the Gammon display corresponding to the starting board setup shown in Figure 1, Section III.

```
*********************
**
**   INITIAL BOARD SETUP
**
**              3   1        3 3        1   3
**  0           X   X        0 X        0   0          X
** |O. . . . .X| .X. . . .0|X. ... .0. |O. . . . .X|
** 24          19 18        13 12         7 6          1
**
**   STRATEGIES:  W: POSITION  B: POSITION
**
*********************
```

Figure 2. Program Representation of the Backgammon Board

3. Moves are represented on this board by specifying the numbers of the points from and to which the men move. A "half move" is a move of a single man the number of points shown on one die, such as 12-17 for a die showing 5. A "single move" involves either one man in two steps, one for each die, or two men, each moving the number of points shown on a single die. This is the normal move. For example, for a roll of 5-2 the move 12-17, 17-19 denotes the move of one man from the 12-point

to the 17-point, and thence from the 17-point to the 19-point. A "double move" involves four die units since a throw of doubles is given twice the amount shown. For a throw of 4-4 for example, a double move might be denoted as 12-16, 12-16, 12-16, 16-20, which represents the move of three men from the 12-point to the 16-point and one man up four points from the 16-point.

From White's standpoint, a move which brings a man on the board from the bar is of the form 0 - P; and a bearing off move as P-25, where P is the number of some point on the board. The role of the 0-point and the 25-point as extensions of the board are reversed, of course, for Black.

C. GAME STATISTICS

Several numeric quantities which describe significant features of the state and progress of the game are maintained by the program. Separate sets are maintained for White and Black.

These statistics are:

1. The current strategy employed by each player.

2. The cumulative dice count. This is a measure of the "resources" allocated to each player.

3. The advance of each player. If we think of the initial board configuration as representing a zero position, then moving a man forward 6 points is an advance of 6. The advance, then, is a measure of a player's progress. By subtracting Black's advance from White's advance we get the relative point advantage for White. This latter number is a key quantity in determining which strategy to select. Note that the advance and cumulative dice count usually become unequal as the game progresses because of men being hit

and sent back, or being prevented from moving by a block.

4. Bear-off Distance. This is a measure of the total number of man-points a player must advance before he can begin the end-game or bearing off phase. From the starting board positions, the bear-off distance is 77. Since the average dice roll provides an advance of approximately 8 1/3 points (doublets give twice as many points as the dice count), the bearing off phase can be reached on average of 9-10 moves, provided no blots are hit.

5. Overlap. This is a measure of the advance required to get one's rearmost men safely past the opponent's rearmost men. It is computed by multiplying the distance (in points) separating the rearmost men by the total number of such men. Overlap is another key factor in strategy selection.

6. The number of home points held. This is a simple tally of the number of White home points held by White, and Black home points held by Black.

7. The number of opponent's home points held. This is a count of White home points held by Black, and vice verse.

8. The number of men left to bear off. The sum of a player's pieces left on the board and those on the bar. The first player to reduce this quantity to zero wins the game.

9. The location of rearmost men. The point on which the rearmost men are resting.

10. The location of the next-to-rearmost men.

11. The number of men on the bar. This number of men must all be brought into play before any other move can be made.

12. The head of the block. The point on which the forward-most men of the block are resting.

13. The tail of the block. The point on which the rearward-most men of the block are resting, or the rear location of the blocking region.

14. The block potential. This number has the form XYZZ, where X is the number of points a player holds in his blocking area, Y is number of points the opponent holds in the same area, and ZZ is a measure of the likelihood of improving the block by filling in the remaining gaps. A perfect block -- six consectively held points -- would have a block potential of 6000, that is, 6 points held, none held by the opponent, and no opportunity for improvement. The blocking potential is used in deciding whether or not to adopt a blocking strategy.

D. THE TACTICS OF GAMMON

Tactics are the fundamental logical elements of Gammon. Each tactic corresponds to some feature or minor objective of the game. Examples of tactics are to advance the rearmost men, to cover blots, to occupy certain key points, to get men into the inner table as soon as possible, to construct a block, and to hit the opponent's blots.

The process within Gammon which selects the next move is solely concerned with generating legal moves which tend to accomplish the objectives of these tactics, and with evaluating moves by measuring the degree to which they achieve the tactical objective. Consider, for example, tactic number 32 which is to occupy the opponent's 7-point. The move generator for tactic 32 finds all legal moves using the current dice roll which result in one or more men moving to the

opponent's 7-point. The move evaluator for tactic 32 examines every move generated and awards points based on the extent to which the move improves the occupation of the 7-point. If the point is already held by you, no points are awarded because the move does not add anything to the existing situation. If, say, the point is unoccupied, moving two men there is awarded more points than moving a single man to the point.

Gammon has the capability to accommodate 50 tactics, of which approximately 45 are active. Some of these are Boolean combinations of others, for example, enter and hit, or enter and cover. Some tactics have operational similarities. For example, tactics 30-39 are all concerned with occupying key points.

A complete list of all tactics with brief descriptions of their objectives is contained in Appendix A.

E. THE STRATEGIES OF GAMMON

Recall that the conventional strategies in Backgammon are position, running, blocking, back, and bearing off. One of the main design objectives of Gammon was to impart the main characteristics of these strategies to sequences of moves, rather than to consider each move separately.

The mechanism developed to implement this concept was to define the strategies as a finite sets of tactics. Each tactic in turn, has an objective which acts as a goal for generating and evaluating moves.

The operational definition of the strategies was arrived at by analysis of the conventional game strategies given in Backgammon literature. The first step was to identify as many tactics as possible,

regardless of the particular style of play with which they were associated. The second step was to group the tactics in such a way as to describe in operational terms the various strategies.

Seven such strategies were ultimately defined. They correspond closely to the traditional strategies. For the running and bearing off strategies, however, two distinct sets of tactics were defined, one describing an offensive style, i.e., the absence of any threat of capture from the opponent, and a defensive counterpart, i.e., the same basic style of play, but conducted in the presence of enemy threat of capture. The seven strategies of Gammon are:

1. Position
2. Running-offensive
3. Running-defensive
4. Blocking
5. Back
6. Bearing off-offensive
7. Bearing off-defensive

Each "machine" player in Gammon is provided with a 7 x 30 array containing the identification numbers of the tactics comprising his strategies.

Within a set, the same tactic may be a member of more than one strategy. For example, tactic 32, Occupy Opponent's 7-point might be a member of the Position strategy, the Blocking strategy, and the Defensive Running strategy. However, analysis of the literature indicates that occupying the 7-point is very important in the blocking game, moderately important in the position game, and of little importance in the running game.

To enable the program to make this kind of distinction, each "machine" player is also provided with a 7 x 30 array of coefficients. The magnitude of each coefficient indicates the relative importance

of the corresponding tactic. These coefficients are used as multiplication factors in the evaluation process. The evaluator for a tactic awards points independently of the strategy. However, this partial score is multiplied by the coefficient from the appropriate strategy and thereby imparts the relative importance of that tactic to the player's current strategy.

In defining the strategies, each tactic was placed in one of three priority classes. The initial coefficient load assigned a value of 4 to those in the highest priority level, 2 in the intermediate level, and 1 in the lower level. As learning progressed these values changed considerably. Gammon was provided with the capability to rearrange the tactic numbers and coefficients within its strategies so as to place the highest valued ones first. As will be pointed out later, this partitioning of the strategy sets was also a useful operational device.

Appendix B contains a listing of a representative set of strategy definitions in terms of the component tactics.

F. MOVE GENERATION AND EVALUATION

The process of generating and evaluating moves is the heart of Gammon. Two of the program modules are directly involved in this process and three others indirectly.

The information required at the beginning of the cycle is the current game board, the dice roll, the identity of the player who is to move (the active player) and his strategy. The logic of the cycle is basically as follows:

1. Retrieve the first tactic identification number from priority level 1 of the appropriate strategy set of the active player.

2. Generate moves for this tactic.

3. Repeat the process for all other tactics in the first priority level.

4. If no moves have been generated for first level tactics, repeat the process for level two tactics, etc.

5. Evaluate each move on the list using the tactics of level one, multiply the partial score of each tactic by the corresponding coefficient and accumulate the total score for the move.

6. If the total score of one of the moves is greater than the score of the other moves by some predetermined margin then select that move. Otherwise resolve the tie by evaluating those moves with nearly equal scores using the tactics at level two, and so on.

Thus the more heavily weighted tactics within a strategy definition will tend to dominate the selection process at both the generation and evaluation stages. This "pruning" of the conceptual game tree was designed primarily to reduce the processing time in order to improve the response time in the interactive version. In the batch mode, where processing time is less critical, this device is overridden by continuing to generate moves until some predetermined maximum number of moves is reached. The criteria for "tied" scores can be adjusted at the terminal, providing another means of varying the number of moves considered.

In assigning partial scores, the gradations of "goodness" were kept within the range of 0-1. Interim scores were assigned on a 64 point scale (e.g., 16 points for a fair move, 32 for good, 64 for perfect), divided by 64, and then multiplied by the coefficient.

All moves are generated and evaluated as if White were to play. On Black's move, the board is converted prior to the cycle and

re-translated on return.

G. RISK EVALUATION

As noted earlier, Gammon does not normally explore the game tree beyond the first level. However, in the event that a proposed move leaves a man exposed, the program assesses the consequences by assigning a risk value to that move. The logic in this assessment is essentially as follows.

1. Locate the exposed man and determine if it has captured an opponent as a result of this move. If so, award positive points depending on the value of the captured man (his advance). Moreover, if it is probable that the captured man will be prevented from coming back on the board because of positions which you block, then award additional points.

2. Scan ahead on the board and locate enemy pieces which may hit your blot on the next play. Subtract points based on the advance of your man, the number of threatening men, and the probability that these men will be able to hit. These calculations are simplified by the fact that probability tables exist in Backgammon literature for (1) entering the board as a function of the number of open points and (2) hitting a blot as a function of its distance away and the number of intervening closed points.

3. Repeat the first two steps for each exposed man on the board, and accumulate the values.

The risk value assigned to a move may be either positive (e.g., when a man has been captured and little threat of a return hit exists) or negative.

Mechanically, risk is treated as a tactic. It has a coefficient which reflects the importance of risk to the particular strategy.

H. STRATEGY SELECTION

The determination of the strategy to be used is determined by analysis of two sets of numbers: the game statistics, described above, and a set of parameters which delimit the transition points from one strategy to another, in terms of the relative point advantage.

Typical advice from Backgammon experts is to use the back game only when one is "far" behind, and the blocking game when one is "slightly" behind. The program uses the parameters to define these points. For example, the transition parameter for the back game is - 50, i.e., the program will not select the back strategy unless the player is at least 50 points behind. Meeting the parameter criteria alone is usually not sufficient to justify a transition.

Another key criterion is the other player's strategy. With rare exceptions, a back game and a blocking game are countered by a running game unless the opportunity for blocking is high.

Selection of bearing off strategies is relatively straightforward, since the rules of the game define exactly when bearing off is permitted. The choice of defensive or offensive styles is simply a matter of whether a threat exists.

These and other considerations are embodied in the set of decision rules which the program uses to select a strategy. If the transition tests fail, the program retains the previous strategy, in order to prevent unnecessary jumping back and forth between strategies.

Two other factors need mention here. First, each player is permitted to "know" the other's strategy. This is largely justified on the basis that in actual play, the strategy of the other person, if he has one, is usually apparent.

The second, and perhaps more valid weakness of the strategy selection process is that the selection is made before the moves are generated, hence consideration of the next move as a factor in the selection is disregarded. The human player normally includes in his analysis an assessment of whether a suitable move exists to justify a choice. The decision to adopt a blocking game, for example, is not made unless a reasonably good blocking move can be made on the next play.

I. THE PROGRAM MODULES

The major functions performed by the program were identified earlier. In this section these functions are associated with the specific modules which accomplish them, and in the following section the general interrelationships among the modules is outlined.

Detailed descriptions of the modules, their principal subroutines, and significant programming techniques are contained in Appendix C.

Briefly, the modules and their major functions are as follows:

1. GAMMON is the designated main program, but acts only as an interface between the terminal user and the rest of the program, accepting two-letter command codes and branching to the appropriate entry point in other modules.

2. SETUP is activated only in the beginning of a game. It retrieves the desired board configuration, strategy and coefficient sets from disk files, and initializes the game statistics.

3. ROLL generates the random integers used as the dice roll.

4. MASK produces a bit array which identifies all positions on the current board which the active player may either move to or move from, using various configurations of moves with the current dice roll.

5. MVGEN contains the move generators for each of the tactics. It refers to the array produced by MASK to ensure that any move generated is legal. To avoid duplication, moves are uniquely encoded using bit strings, each of which records all generated moves of a certain type or geometrical profile.

6. MVLIST decodes the bit strings encoded by MVGEN. The moves are translated into the customary notation and placed in a conventional list.

7. MVEVAL contains the evaluators for each tactic. In performing the evaluations of each move on the list, it implements each proposed move on a temporary board, analyzes the changes, and measures the degree to which the objective of each appropriate tactic is achieved as a result of the change. Risk analysis is also performed by this module.

8. MVPICK places the move with the largest score at the top of the list and rearranges the remaining list such that all moves whose scores are within a specified interval of the largest are placed next on the list. Those not qualifying are discarded.

9. MVIMP implements the move at the top of the list after the evaluation cycle is completed. It updates the current game board, the game statistics, and certain data used by other modules.

10. PUT1 displays the updated board, the move and its score, the number of men on the bar, and the current strategies of each player. The Computer Output section of this report contains a complete game using the standard format. An abbreviated format is used in terminal play in order to reduce printing time.

11. ADMIN is the most complex module. It performs the "main program" function, i.e., calls the other modules in the appropriate sequence, and arranges data for their use. Its subroutines determine the strategy to be used, check the legality of moves entered at the terminal, and execute most of the commands issued by the terminal user. ADMIN detects certain game situations which require a departure from the logic of the main cycle, such as entering a single man from the bar, end-of-game, and no-moves.

J. GENERAL PROGRAM FLOW AND LOGIC

Having examined the concepts and components of Gammon, the general flow of the game can now be summarized. Figure 3 is a Flow chart of the overall logic of the program. The chart applies to a single batch game between two virtual machine opponents.

Two branch points are noted on the chart which require elaboration. In both cases a deviation from the normal flow of the program is required in order to handle a unique situation. Each case requires a single move to be constructed from two separately generated half-moves. The first special branch point is DO_ENTER. This part of the program deals with the situation where a player has a single man on the bar which must be brought into play immediately. The first half-move, if available, brings the man on the board. An interim updating is then accomplished. The next step is to generate the

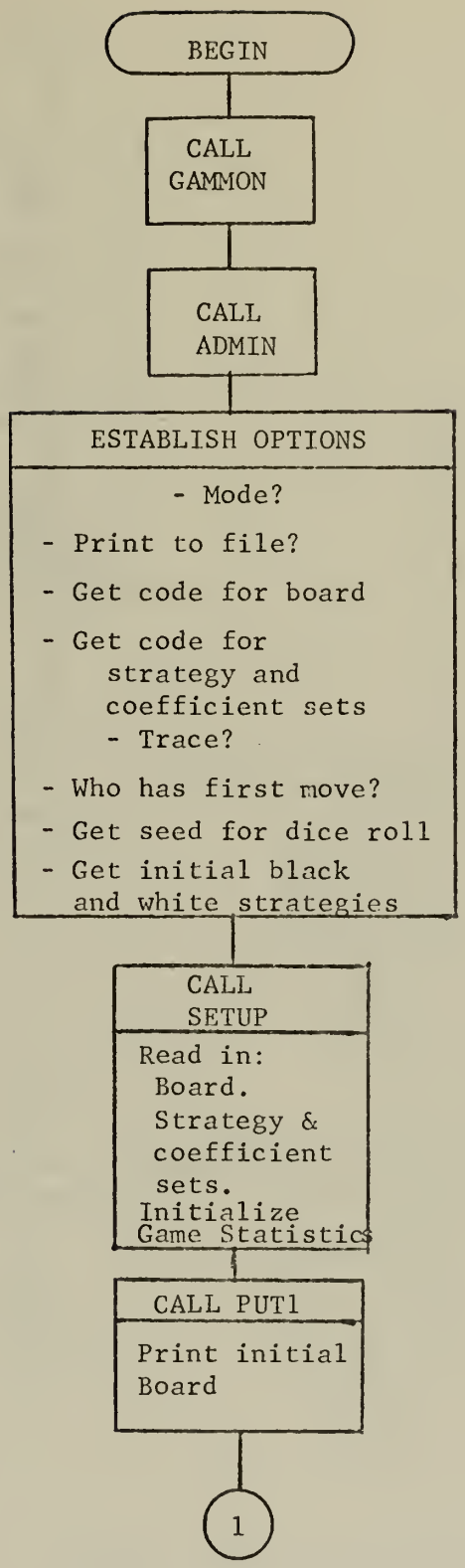

Figure 3a. General Program Flow

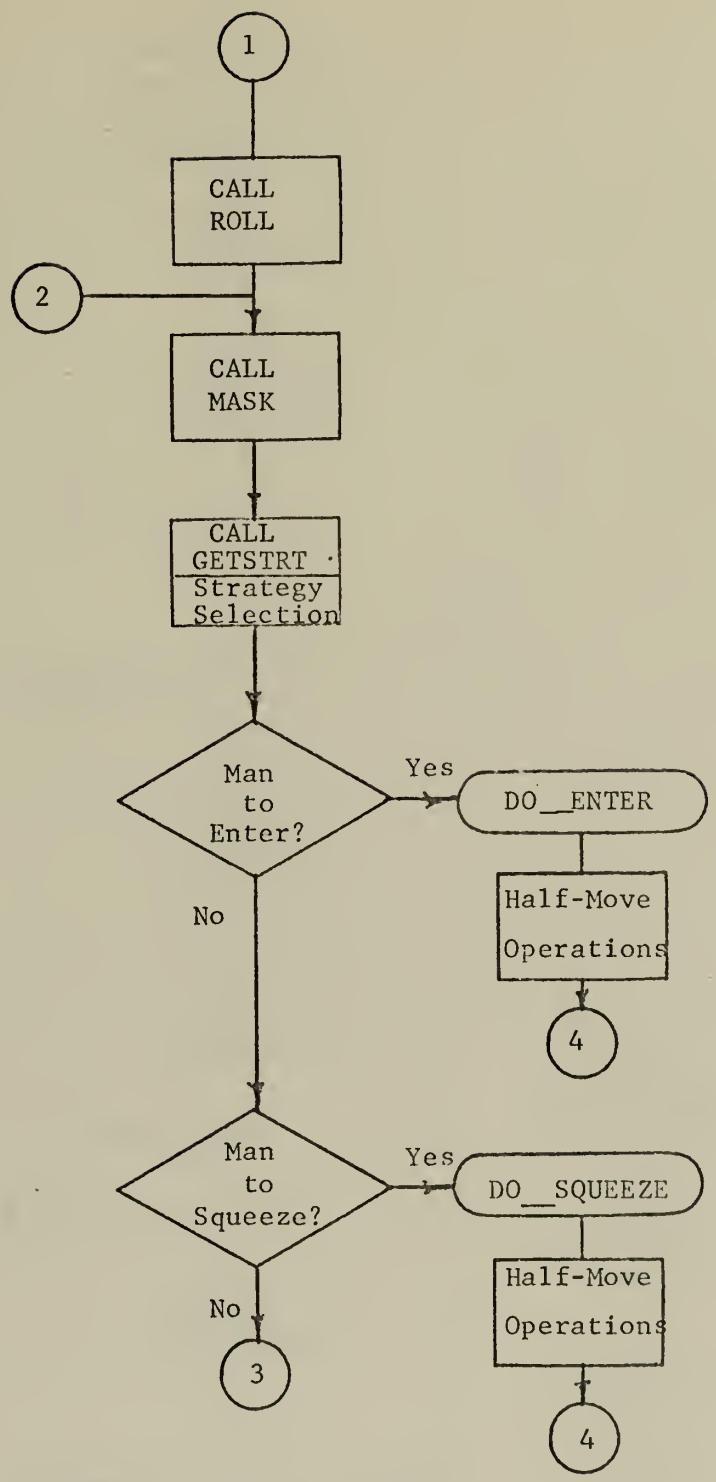

Figure 3b. General Program Flow (cont.)

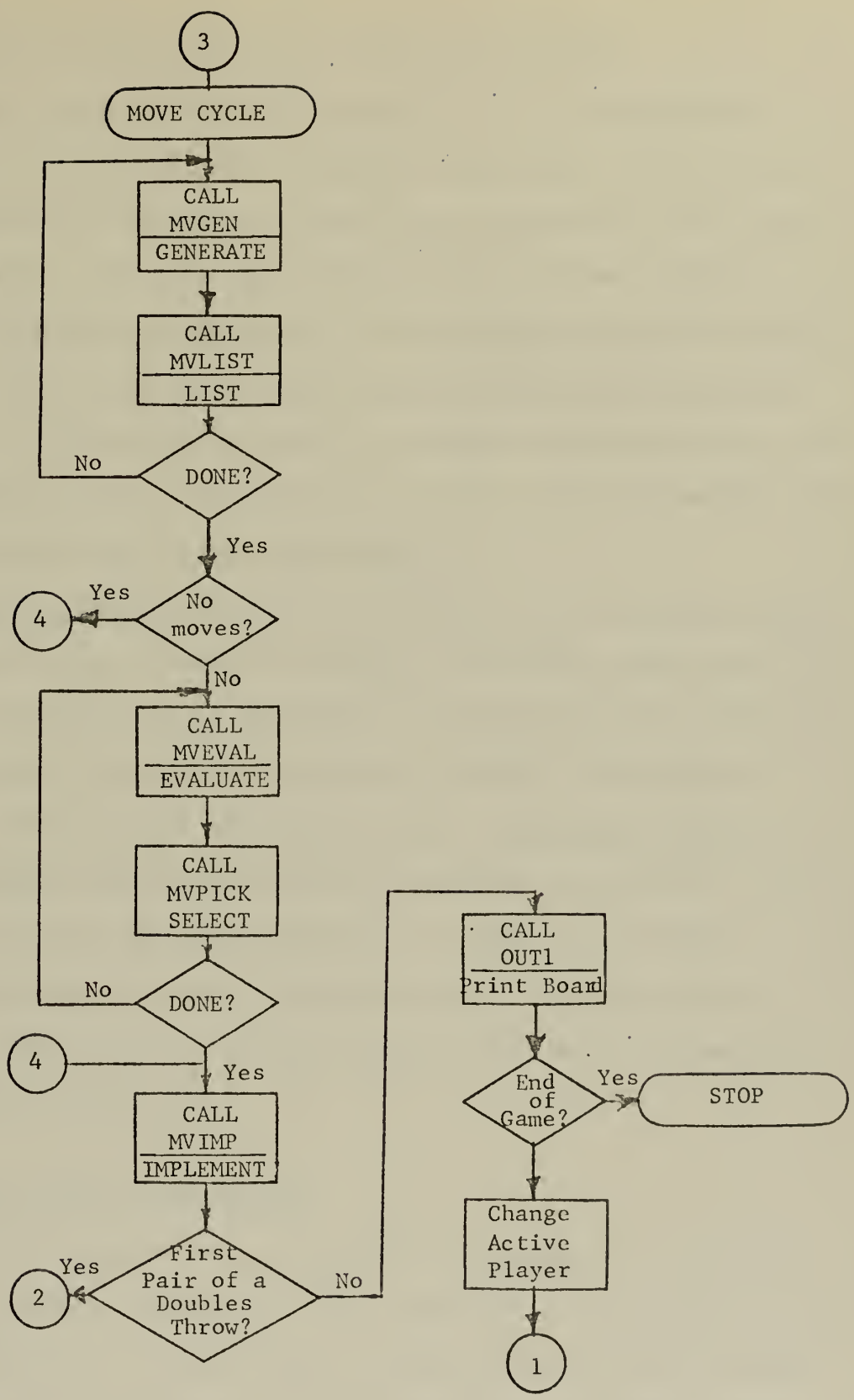

Figure 3c. General Program Flow (cont.)

second half of the move using the other available die.

The second special case is performed in the DO_SQUEEZE branch. This situation occurs when a player has a single man left to bring into the bearing-off area, and it is possible to do so using only one die. The first half-move attempts to squeeze this last runner into the bearing-off area. If this is successful, the strategy of the player is changed to bearing-off, and the board is partially updated. The second half-move is a normal bearing-off move using the other die. The two half-moves are then paired up and the entire move implemented in the usual manner.

Gammon departs from the established rules of Backgammon in only one respect, the play of doublets. According to the rules, all four half-moves are accomplished as a single sequence. However, mechanisms to analyze and record all possible configurations of 4-unit moves become extremely complex. Gammon compromises by implementing a doubles move in two single-move steps. On the first of the two, the risk evaluator is disenabled. The effect is to give the system a chance to accept a higher risk move on the first cycle, knowing that it will have an opportunity to make adjustments immediately afterward.

K. LEARNING MECHANISMS

The learning technique employed in Gammon was a relatively simple deductive mechanism. The coefficients used in the evaluation polynomial were initially loaded by the programmer and remained unchanged during run time, except upon command by the operator.

The following devices were available to the human "tutor" in order to induce learning.

1. Following the selection of a move by the program, the tutor may assign a numerical grade to the move. The program repeats the evaluation process and stores the partial scores for each tactic involved. A partial score is the product of the evaluation score for that tactic and the corresponding coefficient. Also accumulated are the sums of all positive partial scores and all negative partial scores. The second evaluation is carried through only as many priority levels as was the original process.

The program then adjusts the coefficients of the tactics making up the current strategy using the following rules:

a. An incremental change, XDELTA, is computed for the coefficient of each tactic which participated in the evaluation process.

b. XDELTA is proportional to the magnitude of the assigned numeric grade.

c. XDELTA is proportional to the ratio of the partial score for that tactic to the partial sum having like sign. Thus, if a given tactic contributes twice as many positive points to the final move score as another tactic, the magnitude of the change in the first coefficient would be twice as large as the second.

d. The sign of XDELTA is positive if the signs of the grade and the partial score are the same, negative if these signs are different. For example, if a move were given a grade of +3, the coefficients of all tactics which helped make the total move score positive (i.e., their partial scores were positive) would be made more positive. The coefficients of those tactics which contributed negatively to this "good" move would be decreased.

e. The signs of coefficients are kept positive. The continuous adverse contribution of a tactic will result in its coefficient becoming increasingly small.

2. A second training device is to carry out the same kind of process described above using: (a) moves generated by the program, but not selected because of an insufficient score, or (b) moves not generated by the program but supplied by the tutor. After the program has completed its move, all or part of the list of moves which it considered may be displayed, along with the move scores. The tutor may select one or more of these moves to be re-evaluated and graded as described above. Moreover, if the program failed to generate moves which the tutor thinks are particularly good or bad, examples of these may be entered, evaluated, and graded.

Following a training session the tutor may invoke routines which will display the current or old sets of tactics or coefficients, or store the current sets on disk files for future use. It is also possible to perform a sort operation on the current coefficient sets, placing the largest coefficients in the first priority level. In this process, the corresponding tactic numbers are similarly rearranged.

L. MODES OF PLAY AND PROGRAM OPTIONS

Gammon is a versatile program with respect to the number and types of different operations which may be performed. Some of these have been mentioned in the course of the foregoing discussion. However, a consolidated summary of these options will better reveal this scope.

1. The program may operate either interactively or in a batch mode.

2. In the batch environment, three different types of runs may be specified. The first is a standard single game, the second is a series of two games with both players being provided identical dice rolls, and the third is a series of five games, each of which is conducted with identical rolls. The technique of using identical rolls for both players during a play-response cycle is to reduce inequities introduced by the random dice rolls.

3. Program output may be directed to a disk file for printing at a later time, or may default to the terminal or the system printer.

4. In the terminal version, the user may request the program to evaluate his moves, providing another means of judging the system's performance.

5. A trace option may be specified. With the trace, the progress of the program throughout all the modules is provided in detail, including abbreviated boards and partial scores for every move evaluated.

6. Any of several starting board configurations may be specified.

7. The user may alter, during play at the terminal, the trace option, the strategy of either player, or the value of the criteria for "tied" moves. A game may also be restarted, without terminating execution.

8. The user may have displayed all or a specified number of the moves in the current move list. He may also enter a move, have the program determine whether it is on its current list, and have the score displayed.

9. The user may have displayed the current or original set of tactics and coefficients. This data may also be directed to a disk file.

10. The user may grade the move selected by the program, any move on the current list, or any move the user chooses to supply.

11. The current set of coefficients may be exchanged between the two players on command.

12. The current board configuration, or sets of tactics or coefficients may be stored on disk on command.

M. ENVIRONMENT

The program was initially developed using the Cambridge Monitor System (CMS) on the IBM 360/67 at the Naval Postgraduate School Computer Facility. In the midst of the debugging process CMS was replaced by IBM's Time Sharing System (TSS) and the program was converted. Considerable difficulty and loss of time accompanied this conversion.

Gammon consists of approximately 2900 PL/I statements and is compiled by the TSS version 5 PL/I compiler into a total of about 100 K bytes of program and storage. In its final form the program comprises 11 modules ranging in size from 25-970 PL/I statements.

In the interactive mode, the time required by the program to complete one play varied from 5 seconds to 2 minutes depending on the number of tasks active in the TSS system and the number of possible moves needing evaluation. The average response time at the terminal for the program to select its next move was approximately 30 seconds. In the batch mode, the average CPU time required to play one complete game was approximately one minute.

V. RESULTS AND CONCLUSIONS

A. PLAY OF STRATEGIES

In general, Gammon meets its design objective with respect to its play of the individual strategies. Moreover, the game strategies as played by Gammon bore a reasonable resemblance to the traditional Backgammon game styles they were intended to emulate. Least acceptable was the play of the blocking and back strategies. To some extent this was anticipated because the successful construction and advance of a block requires considerable skill, even for human players.

The play of most strategies was uneven or spotty. The program would frequently select a move consistent with the overall characteristics of the strategy, but quite unsuited for the particular situation.

At times, the move selected seemed overly conservative with respect to the general intent of the current strategy. On other occasions the move was overly aggressive. When faced with a block, for example, Gammon fails to capitalize on opportunities to move through or over the block.

The risk evaluator was adequate provided the major consequences of a move were restricted to the move and the immediate reply. Moves selected because of favorable immediate consequences (e.g., a large positive point award resulting from a capture) often had quite unfavorable consequences two moves later.

B. STRATEGY SELECTION

The mechanism for strategy selection performed at an acceptable

level. However, since this mechanism only required the application of pre-defined decision rules it cannot claim any special distinction. Because the logic of the process favored retention of the current strategy (unless the tests for transition were met) the program occasionally persisted with a strategy beyond the point suggested by common sense. A case in point is blocking. The program usually continued with the blocking strategy even though the conditions for maintaining an effective block no longer existed. This defect could be corrected by refining the decision rules.

C. LEARNING

A thorough evaluation of the learning mechanisms was not possible because of time constraints.

Based on the training that could be accomplished, the learning mechanism did improve the program's performance. In particular, the consistency of play was improved, probably because a better balance was achieved among the coefficients. Little improvement was noted with regard to the program's ability to exploit unusually good opportunities, or to protect itself properly when presented with an unusually high threat situation.

The learning process, because it requires human participation, is quite slow. The time to complete a reasonable session exceeded one hour. Moreover, the frequent occurrence of system failures lowered the probability of successfully completing any extended training session.

Repeated gradings after one play would sometimes disproportionately alter a small number of coefficients. As a result, these inflated values would unfavorably influence the evaluation at a later

stage of play. Thus, the training session needed to be balanced throughout the course of a game. As noted above, it was difficult to accomplish such prolonged sessions.

D. MAJOR AREAS OF DIFFICULTY

In the development stages, the most difficult design problems centered around the techniques for determining the legality of moves, the encoding and decoding procedures, the provisions for handling entering and bearing off moves, and the compatibility and coordination among the modules.

Whenever possible, modules were fully tested independently of the others. However, many of the major program functions could not be adequately checked out until all of the modules were brought together. The main difficulties that arose at this time centered around the various special situations requiring deviations from the logic of the main cycle. Half-move operations, occurrences of no moves, doublets, and simultaneous provisions for the various modes of play and options were major complications.

E. MAJOR STRENGTHS AND WEAKNESSES

The major strengths of the program may be summarized as follows:

1. In general, the program met its design objectives with respect to the play of the individual strategies.

2. The programs play an interesting, but not particularly skillful game of Backgammon.

3. The response time is small enough to be able to complete a game in one setting. When the number of users in the system is

low, the response time approaches that of human play.

4. The program is quite versatile, permitting the terminal user to select from a wide range of options, and alter conditions during execution.

The major weaknesses are:

1. In retrospect, some parts of the program are unnecessarily complex. Some techniques were developed to minimize problems which turned out to be of less importance than originally thought.

2. The program does not react properly to unusual temporary game situations requiring departures from standard styles of play.

3. The play of the strategies tended to be uneven and spotty.

4. The lack of a dynamic learning mechanism which could operate in batch mode seriously impedes the rate of learning. This deficiency stems from the fact the program has no standard against which to judge the adequacies of its strategies, except the human tutor.

F. SUGGESTED IMPROVEMENTS

To alleviate some of the deficiencies noted above the following are considered to have the potential to produce detectable improvements with a reasonable investment of effort.

1. The logic of the move generators could be refined. At present, a move generator essentially generates all possible moves which might satisfy the objective of that tactic. When many tactics are involved, the final list actually becomes non-selective. Another way of looking at it is that since there are so many tactics, practically every legal move is bound to satisfy one of them. Nevertheless, most of the generators could be made more specific, both with respect

to (a) producing moves with higher positive potential, and (b) screening out moves with higher negative potential, or those more clearly unsuited for the particular situation.

2. The evaluators could undergo a similar refinement, if only to eliminate inconsistencies introduced because of the author's lack of Backgammon expertise.

3. The program could be provided with the capability to alter dynamically the composition of its strategy sets. Deletions could be accomplished by the program, based on coefficients which become increasingly small. New tactics could be selected at random from a list provided by the programmer, or by a mechanism such as that described below.

4. The program needs a capability for accommodating short range objectives. The current strategies are general in nature and are not well suited for exploitation of more pressing short range goals. Many of these situations could be described as generalized board configurations. They might have the form of templates which could be matched against the present or proposed board in order to determine whether or not the situation exists. If a match were made, subsequent generation and evaluation procedures would attempt to satisfy the objective. However, providing the program with the capability to recognize and describe these short range objectives would not be a simple task.

G. CONCLUSIONS

The main conclusion of the study is that the validity of the strategy concept developed with Gammon was generally verified by the performance of the program. The play of Gammon using its strategies took

on the overall characteristics of the corresponding Backgammon styles of play. However, the concept needs to be modified in order to better recognize and accomplish short range objectives, particularly high-opportunity or high-threat situations.

APPENDIX A

THE TACTICS OF GAMMON

TACTIC NO.	NAME/DESCRIPTION
1	BRING UP REARMOST MEN
2	CLOSE YOUR INNER TABLE
3	CLOSE OPPONENT'S INNER TABLE
4	DO NOT EXPOSE BLOTS
5	HOLD AS MANY POINTS AS POSSIBLE
6	BUILD A BLOCK
7	DEPLOY CATCHERS (BLOTS THAT ONE WANTS HIT IN ORDER TO BRING ON IN THE BACK BLOCK AREA)
8	EXPOSE BLOTS IN YOUR INNER TABLE
9	FREEZE RUNNERS
10	MOVE MEN TO YOUR INNER TABLE
11	BRING MEN ON 8-POINT HOME IN PREFERENCE TO MEN ON 7-POINT
12	ADVANCE THE BLOCK
13	MAXIMIZE SAFE MOVES
14	LEAVE A LONG RUNNER BACK
15	UNUSED
16	MOVE MEN UP FROM THE COMFORT STATION (OPPONENT'S 12-POINT)
17	BEAR OFF AS MANY MEN AS POSSIBLE
18	MAXIMIZE PROBABILITY OF BEARING OFF TWO MEN ON THE NEXT PLAY
19	WHEN BEARING OFF DEFENSIVELY, MAKE HIGH TWO POINTS BOTH EVEN OR ODD

20	WHEN BEARING OFF, MOVE TO LOWER POINTS
21	COVER BLOTS
22	DISTRIBUTE MEN IN OUTER TABLES
23	BRING MEN INTO SAFE AREA BEHIND OPPONENTS REARMOST MEN
24-29	UNUSED
30	OCCUPY OPPONENT'S 5 POINT
31	OCCUPY OPPONENT'S 4 POINT
32	OCCUPY OPPONENT'S 7 POINT
33-39	OCCUPY YOUR 3-9 POINTS
40	HIT ALL BLOTS
41	HIT BLOTS IN OUTER TABLES
42	HIT BLOTS IN OPPONENT'S INNER TABLE
43	HIT BLOTS IN YOUR INNER TABLE
44	UNUSED
45	ENTER ON LOW POINTS
46	ENTER AND HIT
47	ENTER AND DO NOT HIT
48	ENTER AND COVER
49	ENTER AND EXPOSE
50	RISK VALUE

APPENDIX B

COMPOSITION OF REPRESENTATIVE STRATEGY SET

STRATEGY	LEVEL 1 TACTICS	LEVEL 2 TACTICS	LEVEL 3 TACTICS
1. (POSITION)	1, 5, 13, 21, 30 32, 35, 37, 48, 50	2, 4, 11, 23, 31 36, 41, 35, 36, 34	6, 10, 12, 33, 38, 39, 43
2. (RUN-OFF)	1, 10, 11, 14, 16, 47	5, 30, 32, 35, 37 43, 45, 50	31, 34, 36
3. (RUN-DEF)	1, 2, 4, 10, 11, 14, 21, 23, 48, 50	5, 13, 16, 30, 32, 35, 37, 41, 45, 46	31, 34, 36
4. (BLOCKING)	6, 12, 45, 34, 35, 36, 37, 38, 48, 50	13, 21, 33, 39, 41, 43, 45, 46	10, 11, 22
5. (BACK)	3, 6, 7, 8, 30 31, 32, 40, 49	9	10, 50
6. (BEAROFF-OFF)	17, 18	50	
7. (BEAROFF-DEF)	17, 18, 19, 20, 50		

APPENDIX C

DESCRIPTIONS OF THE PROGRAM MODULES

1. Gammon

This module is the designated main program. However, in the batch version, upon receiving control it immediately passes it to ADMIN, which actually performs the main program function. In the interactive version control is passed back to GAMMON after each move or after certain types of errors. The principal function of GAMMON is to accept commands from the operator and branch to the appropriate point. Several two-character command codes may be issued at the terminal. The purpose of the commands are summarized below.

GO -- continue the game.
XX -- stop the game
RE -- restart the game from the beginning.
GR -- grade a move.
TR -- change the trace option.
CS -- change a player's strategy.
CD -- change the quantity "Delta" which defines the range within which evaluated move scores are considered equivalent.
EV -- enter a move of the operator's choice, have it evaluated and graded.
GE -- generate a move for some other strategy.
DI -- display the present or old set of tactics or coefficients.
SO -- sort the present sets of tactics into descending order according to the corresponding coefficients.
LO -- locate the given move in the list of moves and display its score.
LI -- display all or portions of the list of generated moves.
SW -- exchange the strategy and coefficient sets between the two players.
ST -- store the present strategy and coefficient sets in disk files.

2. SETUP

This module is called only at the beginning of a game. It receives as arguments the file codes of the initial board configuration and the strategy and coefficient sets of the players. It reads in this data and initializes the game statistics. It also reads in the tables containing the probabilities of hit and entry, as well as a table containing recommended opening moves.

3. ROLL

ROLL returns a pair of integers on the interval (1, 6). After generating these numbers, which serve as the dice roll, it passes them back such that the first die is greater than or equal to the second. ROLL is initially called with a positive odd integer SEED. Thereafter, it is called with an argument of 0. The series of pseudo-random numbers may be reproduced by providing the same seed.

4. MASK

The function of this module is to provide a reference by which other modules can determine whether or not a legal move can be made either to or from a given point, MASK requires as arguments the current game board and dice roll (D1, D2). In addition to these arguments, MASK is provided two bit strings of length 24. Each bit position corresponds to a point on the board. One string, TO_STR, has a 1 in each position corresponding to a point on the board which is not held by an opponent. Thus it records every position to which the active player may legally move, regardless of the dice roll. The second string, FR_STR, contains a 1 in those positions corresponding to points on the board where the active player has men, i.e., positions

from which the active player may move, regardless of the dice roll. MASK then proceeds to construct a bit array indicating the legal to and from positions for the current dice roll. The criteria for moving a man up D1 points from point P is that bit P of FR_STR =1 and that bit (P + D1) of TO_STR =1. This is done for all four possible configurations of moves involving a single man; (1) moving D1 points; (2) moving D2 points; (3) moving D1 points and thence moving D2 points (a D1-D2 "bounce"); and (4) a D2-D1 bounce. The TO and FROM arrays have dimensions 4 x 24 where the columns correspond to the points on the board and the rows indicate the type of move.

The construction of the arrays is accomplished by performing sequences of shift operations (in units of D1 and D2) and Boolean AND operations.

Suppose that a generator for a tactic needs to know whether a man can be moved to a certain point on the board. By examining the appropriate column of the TO matrix, it can determine not only whether any legal moves exist, but all possible forms of such moves.

5. MVGEN

This module contains the generators for the tactics. The general logic of the move generation process was discussed above. Here, the method by which moves are recorded will be described.

One approach to recording moves would be to generate all moves, record them in the X-Y, W-Z notation and then analyze the list to remove duplicates. This method, however, would envolve considerable processing time just for searching. The same can be said for the method of searching for duplicates after each single move was generated. To minimize this problem, Gammon records its generated

moves in coded bit strings such that duplicate moves simply set the same bits equal to 1. After the generation process is completed, the bit strings are decoded into the conventional X-Y, W-Z notation.

The coding process requires a total of 17 bit strings, seven of length 24 to record all possible configurations of normal single moves, eight of length 6 to record all possible configuration of bearing off moves, and two of length 2 to record entering moves.

"Normal" moves have seven possible configurations. Their descriptions and the names of the strings used to record them are:

1. SH1 - move a single man from point P to point P + D1 (single hop on D1).

2. SH2 - a single hop on D2.

3. DH - moving two men on a throw of doubles from point P to point P + D1.

4. B12 - a D1-D2 bounce.

5. B21 - a D2-D1 bounce.

6. CNV - a converging move, i.e., one in which men from different points move to the same point.

7. FRK - a forking move, i.e., one in which two men from the same position move to two different points.

A last, and usually quite large, class of moves are those involving all combinations of single hops, i.e., each single hop recorded in SH1 paired with each single hop recorded in SH2. These combinations are implicit in the SH1 and SH2 strings and are combined when the strings are decoded.

MVGEN uses three subroutines to set the proper bits in these strings.

Each has two arguments -- a point on the board, and the number of die units, 1 or 2, available as resources to accomplish the move.

The subroutines are (1) MVTO, which encodes all moves resulting in a man or men being placed on the specified point; (2) MVFM, which encodes all moves resulting in a man or men moving from the specified point, and (3) MVHIT, which encodes all moves resulting in hitting an exposed enemy piece located at the specified point.

Usually, the generator for a tactic performs a preliminary analysis to determine the points on the board which should be involved in the move, and then calls the appropriate subroutines.

6. MVLIST

The function of this module is to decode the bit strings, translating the moves into conventional notation, and placing the moves in a list. MVLIST requires as arguments the encoded bit strings, the dice roll, and the availability number (1 or 2) which indicates whether to decode only half-moves or regular single moves. MVLIST proceeds through each bit string, noting the location of 1's and performing the translation.

Suppose, for example, that bit 20 of FRK (fork move) were equal to 1. Then, assuming a dice roll of 5-2, the move 20-15, 20-18 would be placed in the list. Note that all moves are treated as if White were to move, hence the movement from higher to smaller point numbers.

The move list is a 50 x 4 array with each row corresponding to a different move, and the four columns indicating the points of the board.

In decoding the single hop strings, MVLIST forms all possible pair-wise combinations after first eliminating those which form a move already encoded as another type.

7. MVEVAL

This module evaluates each of the moves in the current move list and assigns points (plus or negative) based on the degree to which the move succeeds (or fails) to accomplish the objectives of the tactics which define the active player's current strategy. After evaluating each tactic it multiplies the partial score by the corresponding coefficient and accumulates the score until the entire process for that move is completed. It then proceeds to the next move.

In making the evaluation, MVEVAL constructs a temporary game board which incorporates the results of the move under evaluation. In many of its assessments it compares the old and new boards to determine change as a result of the move. MVEVAL performs its evaluations one priority level at a time. If further evaluations are required it repeats the process at the next level for those moves still in contention.

Recall that in the terminal version the human "tutor" can assign a grade to the selected move. In order to adjust its coefficients the program needs to know the partial scores for each tactic evaluated. However, it was considered wasteful of storage to retain all of these scores for all moves. Hence, when a move is given a grade, MVEVAL repeats its evaluation process, at which time it saves all of its partial scores for the move, as well as accumulating the sums of all positive and negative partial scores. The use of these special sums will be discussed later.

8. MVPICK

Following the evaluation of the list of moves, MVPICK locates the move with the maximum score and places it at the top of the list.

Immediately below this move are placed all moves whose scores are within distance DELTA from the highest-scored move. The remainder of the list is discarded. By making DELTA large we carry over more moves from one priority-level evaluation to the next, thus reducing the possibility that an excellent move will be eliminated from further consideration only because it scored poorly at one priority level. On the other hand, by making DELTA small, we quickly reduce the size of the list, thereby conserving processing time, and in the case of the interactive version, improving the response time.

If, at the end of three evaluation cycles (one for each priority level) there is more than one move on the list, the first move is selected for implementation.

9. MVIMP

Implementing the selected move is the function of this module. If the active player is Black it translates the move from the White-notation in which it was provided into the proper terms for Black. It then updates the game board by adding and subtracting men from the appropriate points. It also performs checks to ensure that the move is legal.

It then proceeds to update the game statistics, revising the cumulative dice count and advance, relocating the rearmost men, etc.

Finally, it updates the bit strings used by MASK.

In the process of initializing the game before play begins, MVIMP may be involked by SETUP in the event that the initial board is other than the standard starting configuration, for example, a training board for bearing-off situations. MVIMP then initializes the game statistics by analyzing the board, reading data, or prompting the user to supply needed information.

10. PUT1

This modules performs the function of writing the resulting board following implementation of the move.

In the terminal version the board format is abbreviated in order to reduce the printing time. This was especially critical using TSS because the PL/I output introduces an unacceptably long delay at the end of each printed line.

In the batch mode, the boards may either be directed to SYSPRINT, the normal print channel, or to a disk file data set in the user's library.

Under TSS, a game can be initiated at the terminal as if it were a batch game and run to completion with the output directed to the private file. This file can then be printed off-line, reducing the turn-around time to a few minutes.

A sample of a complete batch mode game is included in the Computer Output section of this report.

11. ADMIN

Left for last is the discussion of ADMIN, the module which is the largest and most complex. It embodies the main logic of the flow and control of the game and, except in name, is the main program. It makes the tests necessary to determine the state and progress of the game, gathers data together for use by the other modules, and issues calls to the other modules in the appropriate sequence. It contains several of the subroutines needed to perform the commands issued by the terminal user.

The principle subroutines used by ADMIN are:

a. GETSTRT.

This routine performs the important function of strategy selection. The major considerations in this process were discussed in Section IV. H.

b. CHECK

The purpose of CHECK is to determine whether a move entered at the terminal is legal. If not, it prompts the user to re-enter the move, after displaying an appropriate diagnostic.

c. STORE performs the function of writing the current board, strategy sets or coefficient sets on disk files.

d. SWAP exchanges the current strategy sets of White and Black.

e. DISPLAY prints the current strategy and coefficient sets of White or Black, along with the initial sets if desired. After a learning session in which the composition of those sets may change, it is useful to be able to display the changes for review and analysis.

f. FIND

It is often interesting to know whether or not a move other than the one finally implemented was generated and what the evaluation score was. FIND performs this service.

g. PUTLIST displays the entire move list, or a specified portion of it.

h. SORTCT restructures a player's strategy and coefficient sets, placing the 10 largest coefficients and the corresponding tactic identification numbers in the first priority level of each strategy, the next 10 largest in the second level, and so on.

In addition to these subroutines, ADMIN contains the code which performs some of the other commands available to the user. Among these are adjusting coefficients as a result of a move being graded, evaluating and then grading in arbitrary move entered at the terminal during a training session, changing the trace option, and changing the DELTA value for move selection.

```
*******************
**
**   INITIAL BOARD SETUP
**
**              3   1        3 3        1   3
**  0           X   X        0 X        0   0          X
** |O. . . . .X| .X. . . .O|X. . . .O. |O. . . . .X|
** 24          19 18        13 12         7 6         1
**
**   STRATEGIES:  W: POSITION  B: POSITION
**
*******************
```

```
*******************
**
**   MOVE NO:   1   ROLL: 6-4    WHITE   24-18 , 24-20
**
**              3   1        3 3        1   3
**              X   X        0 X        0   0          X
** | . . . . .O.X|O.X. . . .O|X. . . .O. |O. . . . .X|
** 24          19 18        13 12         7 6         1
**
**   STRATEGIES:  W: POSITION  B: POSITION
**
*******************
```

```
*******************
**
**   MOVE NO:   2   ROLL: 6-6    BLACK   12-18 , 12-18 , 12-18 , 18-24
**
**              3   1        3          1   3
**              X X X        0 X        0   0          X   BAR: W:  1  B: 0
** |X. . . .O.X|X.X. . . .O|X. . . .O. |O. . . . .X|  MOVE SCORE:        8.76562
** 24          19 18        13 12         7 6         1
**
**
*******************
```

```
*******************
**
**   MOVE NO:   3   ROLL: 6-2    WHITE   25-23 , 13- 7
**
**             3   1        2          1   3
**             X   X  X     0  X       0   0          X
** |X.O. . .O.X|X.X. . . .O|X. . . .O.O|O. . . . .X|  MOVE SCORE:        3.07278
** 24         19  18       13  12        7   6        1
**
**   STRATEGIES:  W: POSITION  B: BLOCKING
**
*******************
```

```
*******************
**
**   MOVE NO:   4   ROLL: 4-2    BLACK    1- 5 ,  5- 7
**
**             3   1        2          1   3
**             X   X  X     0  X       0   0                BAR: W:  1  B: 0
** |X.O. . .O.X|X.X. . . .O|X. . . .O.X|O. . . . .X|  MOVE SCORE:        7.30200
** 24         19  18       13  12        7   6        1
**
**   STRATEGIES:  W: POSITION  B: BLOCKING
**
*******************
```

```
*******************
**
**   MOVE NO:   5   ROLL: 6-5    WHITE   25-20 , 20-14
**
**             3   1        2          1   3
**             X   X  X     0  X       0   0
** |X.O. . .O.X|X.X. . .O.O|X. . . .O.X|O. . . . .X|  MOVE SCORE:        2.40606
** 24         19  18       13  12        7   6        1
**
**   STRATEGIES:  W: POSITION  B: BLOCKING
**
*******************
```

```
*******************
**
**   MOVE NO:   6   ROLL: 3-1    BLACK   17-20 , 19-20
**
**              2            2          1   3
**            X X X X        O X        O   O                BAR: W:  1  B: 0
** |X.O. . .X.X|X.X. . .O.O|X. . . .O.X|O. . . . .X|  MOVE SCORE:        6.77069
** 24         19 18        13 12        7 6          1
**
**   STRATEGIES:  W: POSITION  B: BLOCKING
**
*******************
```

```
*******************
**
**   MOVE NO:   7   ROLL: 6-1    WHITE   25-24 , 14- 8
**
**              2            2          2   3
**            X X X X        O X        O   O                BAR: W:  0  B: 1
** |O.O. . .X.X|X.X. . . .O|X. . . .O.X|O. . . . .X|  MOVE SCORE:        7.65606
** 24         19 18        13 12        7 6          1
**
**   STRATEGIES:  W: POSITION  B: BLOCKING
**
*******************
```

```
*******************
**
**   MOVE NO:   8   ROLL: 6-4    BLACK    0- 4 ,  1- 7
**
**              2            2          2   3
**            X X X X        O X        O X O
** |O.O. . .X.X|X.X. . . .O|X. . . .O.X|O. .X. . . |  MOVE SCORE:        4.40606
** 24         19 18        13 12        7 6          1
**
**   STRATEGIES:  W: POSITION  B: BLOCKING
**
*******************
```

```
*******************
**
**   MOVE NO:   9   ROLL: 5-2    WHITE    8- 3 , 24-22
**
**             2            2          1   3
**           X X X X        O X        O X O
** | .O.O. .X.X|X.X. . . .O|X. . . .O.X|O. .X.O. . |  MOVE SCORE:       11.08319
** 24          19 18        13 12        7 6          1
**
**   STRATEGIES:  W: POSITION  B: BLOCKING
**
*******************
```

```
*******************
**
**   MOVE NO:  10   ROLL: 5-2    BLACK    7-12 , 12-14
**
**             2            2          1   3
**           X X X X        O X        O   O
** | .O.O. .X.X|X.X. . .X.O|X. . . .O.X|O. .X.O. . |  MOVE SCORE:        2.02069
** 24          19 18        13 12        7 6          1
**
**   STRATEGIES:  W: POSITION  B: BLOCKING
**
*******************
```

```
*******************
**
**   MOVE NO:  11   ROLL: 5-5    WHITE    8- 3 ,  6- 1 ,  6- 1 ,  6- 1
**
**             2            2                        1
**           X X X X        O X        O   O     O   O
** | .O.O. .X.X|X.X. . .X.O|X. . . .O.X|O. .X.O. .O|  MOVE SCORE:       15.76544
** 24          19 18        13 12        7 6          1
**
**   STRATEGIES:  W: POSITION  B: BLOCKING
**
*******************
```

```
*******************
**
**   MOVE NO:  12   ROLL: 6-6    BLACK   12-18 , 14-20 , 12-18 , 18-24
**
**          1 2 1          2                        1
**          X X X X        0          0    0    0   0
** |X.O.O. .X.X|X.X. . . .O| . . . .O.X|O. .X.O. .O|  MOVE SCORE:        4.53631
** 24         19 18        13 12         7 6         1
**
**   STRATEGIES:  W: POSITION  B: BLOCKING
**
*******************
```

```
*******************
**
**   MOVE NO:  13   ROLL: 6-1    WHITE   13- 7 ,  7- 6
**
**          1 2 1          1             1          1
**          X X X X        0          0  0    0   0   BAR: W:  0  B: 1
** |X.O.O. .X.X|X.X. . . .O| . . . .O. |O. .X.O. .O|  MOVE SCORE:        7.40606
** 24         19 18        13 12         7 6         1
**
**   STRATEGIES:  W: POSITION  B: BLOCKING
**
*******************
```

```
*******************
**
**   MOVE NO:  14   ROLL: 2-2    BLACK    0- 2 , 19-21 , 18-20 , 19-21
**
**          2              1             1          1
**        X X X X X        0          0  0    0   0
** |X.O.O.X.X.X|X.X. . . .O| . . . .O. |O. .X.O.X.O|  MOVE SCORE:        9.68737
** 24         19 18        13 12         7 6         1
**
**   STRATEGIES:  W: POSITION  B: BLOCKING
**
*******************
```

```
*******************
**
**   MOVE NO:  15   ROLL: 4-3    WHITE   13- 9 ,  9- 6
**
**          2                            2         1
**        X X X X X        O          O   O     O   O
** |X.O.O.X.X.X|X.X. . . .O| . . . .O. |O. .X.O.X.O|  MOVE SCORE:        6.40606
** 24         19 18       13 12         7 6         1
**
**   STRATEGIES:  W: POSITION  B: BLOCKING
**
*******************
```

```
*******************
**
**   MOVE NO:  16   ROLL: 4-4    BLACK   20-24 , 20-24 , 17-21 , 17-21
**
**  1      2                            2         1
**  X      X X X X          O          O   O     O   O
** |X.O.O.X.X.X|X. . . . .O| . . . .O. |O. .X.O.X.O|  MOVE SCORE:        6.35928
** 24         19 18       13 12         7 6         1
**
**   STRATEGIES:  W: POSITION  B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  17   ROLL: 3-3    WHITE    8- 5 ,  8- 5 ,  6- 3 ,  6- 3
**
**  1      2                                2   1
**  X      X X X X          O            O O   O   O
** |X.O.O.X.X.X|X. . . . .O| . . . . . |O.O.X.O.X.O|  MOVE SCORE:       14.32803
** 24         19 18       13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  18   ROLL: 3-2    BLACK    2- 4 ,  4- 7
**
**  1      2                                  2   1
**  X      X X X X          O               O O   O   O
** |X.O.O.X.X.X|X. . . . .O| . . . . .X|O.O.X.O. .O|  MOVE SCORE:        4.32803
** 24         19 18        13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  19   ROLL: 2-1    WHITE    3- 1 ,  3- 2
**
**  1      2                                      2
**  X      X X X X          O               O O   O   O
** |X.O.O.X.X.X|X. . . . .O| . . . . .X|O.O.X.O.O.O|  MOVE SCORE:        6.57284
** 24         19 18        13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  20   ROLL: 6-3    BLACK    4-10 , 21-24
**
**  2      1                                      2
**  X      X X X X          O               O O   O   O
** |X.O.O.X.X.X|X. . . . .O| . .X. . .X|O.O. .O.O.O|  MOVE SCORE:        5.89056
** 24         19 18        13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  21   ROLL: 3-2    WHITE    6- 4 ,  4- 1
**
**  2      1                                        3
**  X      X X X X          O                  O   O   O
** |X.O.O.X.X.X|X. . . . .O| . .X. . .X|O.O. .O.O.O|  MOVE SCORE:        4.97650
** 24         19 18       13 12        7 6          1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  22   ROLL: 5-1    BLACK   10-15 , 18-19
**
**  2      1   1                                    3
**  X      X X X            O                  O   O   O
** |X.O.O.X.X.X|X. . .X. .O| . . . . .X|O.O. .O.O.O|  MOVE SCORE:        5.55203
** 24         19 18       13 12        7 6          1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  23   ROLL: 6-4    WHITE   23-17 , 22-18
**
**  2      1   1                                    3
**  X      X X X            O                  O   O   O   BAR: W:  0  B: 1
** |X. . .X.X.X|O.O. .X. .O| . . . . .X|O.O. .O.O.O|  MOVE SCORE:       10.39450
** 24         19 18       13 12        7 6          1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  24   ROLL: 2-1    BLACK    0- 2 , 15-16
**
**  2      1   1                                  3
**  X      X X X                0                 0   0   0   BAR: W:  1  B: 0
** |X. . .X.X.X|0.0.X. . .0| . . . . .X|0.0. .0.X.0|  MOVE SCORE:        5.91140
** 24         19 18        13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  25    ROLL: 5-4    WHITE    NO MOVES
**
**  2      1   1                                  3
**  X      X X X                0                 0   0   0   BAR: W:  1  B: 0
** |X. . .X.X.X|0.0.X. . .0| . . . . .X|0.0. .0.X.0|
** 24         19 18        13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  26   ROLL: 3-3    BLACK   16-19 , 21-24 , 19-22 , 19-22
**
**  3                                               3
**  X    X X X X                 0                 0   0   0   BAR: W:  1  B: 0
** |X. .X.X.X.X|0.0. . . .0| . . . . .X|0.0. .0.X.0|  MOVE SCORE:       13.39053
** 24         19 18        13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  27    ROLL: 4-1    WHITE    NO MOVES
**
**  3                                          3
**  X   X X X X           O                 O   O   O   BAR: W:  1  B: 0
** |X. .X.X.X.X|O.O. . . .O| . . . . .X|O.O. .O.X.O|
** 24         19 18      13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  28   ROLL: 3-1    BLACK   20-23 , 23-24
**
**  4                                          3
**  X   X X   X           O                 O   O   O   BAR: W:  1  B: 0
** |X. .X.X.X.X|O.O. . . .O| . . . . .X|O.O. .O.X.O|  MOVE SCORE:        2.30722
** 24         19 18      13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  29    ROLL: 4-3    WHITE    NO MOVES
**
**  4                                          3
**  X   X X   X           O                 O   O   O   BAR: W:  1  B: 0
** |X. .X.X.X.X|O.O. . . .O| . . . . .X|O.O. .O.X.O|
** 24         19 18      13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  30   ROLL: 3-2    BLACK   19-22 ,  2- 4
**
**  4   1                                   3
**  X   X X                 O           O   O   O   BAR: W:  1  B: 0
** |X. .X.X.X.X|O.O. . . .O| . . . . .X|O.O.X.O. .O|  MOVE SCORE:        3.57806
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  31   ROLL: 3-2    WHITE   25-23 , 23-20
**
**  4   1                                   3
**  X   X X                 O           O   O   O   BAR: W:  0  B: 1
** |X. .X.X.O.X|O.O. . . .O| . . . . .X|O.O.X.O. .O|  MOVE SCORE:        5.51950
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  32   ROLL: 4-4    BLACK    0- 4 ,  4- 8 ,  4- 8 , 19-23
**
**  4   1                                   3
**  X   X X                 O           X   O   O   O
** |X.X.X.X.O. |O.O. . . .O| . . . .X.X|O.O. .O. .O|  MOVE SCORE:       12.41400
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  33   ROLL: 5-4    WHITE   13- 9 ,  9- 4
**
**  4   1                                       3
**  X   X X                           X     O   O   O
** |X.X.X.X.O. |O.O. . . .O| . . . .X.X|O.O.O.O. .O|  MOVE SCORE:        4.67968
** 24         19 18       13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  34   ROLL: 2-1    BLACK    8-10 ,  8- 9
**
**  4   1                                       3
**  X   X X                                 O   O   O
** |X.X.X.X.O. |O.O. . . .O| . .X.X. .X|O.O.O.O. .O|  MOVE SCORE:        7.80856
** 24         19 18       13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  35   ROLL: 5-5    WHITE   17-12 , 12- 7 , 20-15 ,  6- 1
**
**  4   1                                       4
**  X   X X                                 O   O   O   BAR: W:  0  B: 1
** |X.X.X.X. . |O. . .O. .O| . .X.X. .O| .O.O.O. .O|  MOVE SCORE:        7.70935
** 24         19 18       13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  36   ROLL: 6-1    BLACK    0- 6 ,  6- 7
**
**  4   1                                          4
**  X   X X                                 O   O  O   BAR: W:  1  B: 0
** |X.X.X.X. . |O. . .O. .O| . .X.X. .X| .O.O.O. .O|  MOVE SCORE:        6.03900
** 24         19 18       13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  37   ROLL: 3-2    WHITE   25-23 , 15-12
**
**  4   1                                          4
**  X   X X                                 O   O  O   BAR: W:  0  B: 1
** |X.O.X.X. . |O. . . . .O|O. .X.X. .X| .O.O.O. .O|  MOVE SCORE:        5.74685
** 24         19 18       13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  38   ROLL: 4-3    BLACK    0- 4 ,  9-12
**
**  4   1                                          4
**  X   X X                                 O   O  O   BAR: W:  2  B: 0
** |X.O.X.X. . |O. . . . .O|X. .X. . .X| .O.X.O. .O|  MOVE SCORE:        6.55853
** 24         19 18       13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  39   ROLL: 6-3    WHITE   25-19
**
**  4   1                                       4
**  X   X X                          0   0      0   BAR: W:  1  B: 0
** |X.O.X.X. .O|O. . . . .O|X. .X. . .X| .O.X.O. .O|  MOVE SCORE:        2.00000
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  40   ROLL: 6-4    BLACK   12-18 ,  4- 8
**
**  4   1                                       4
**  X   X X                          0   0      0   BAR: W:  2  B: 0
** |X.O.X.X. .O|X. . . . .O| . .X. .X.X| .O. .O. .O|  MOVE SCORE:        5.55856
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  41    ROLL: 3-3    WHITE    NO MOVES
**
**  4   1                                       4
**  X   X X                          0   0      0   BAR: W:  2  B: 0
** |X.O.X.X. .O|X. . . . .O| . .X. .X.X| .O. .O. .O|
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  42   ROLL: 3-1    BLACK    8-11 , 18-19
**
**  4   1                                          4
**  X   X X                                 O   O  O   BAR: W:  3  B: 0
** |X.O.X.X. .X| . . . . .O| .X.X. . .X| .O. .O. .O|  MOVE SCORE:        5.93356
** 24         19 18       13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  43   ROLL: 6-6    WHITE   25-19 , 25-19 , 25-19 , 19-13
**
**  4   1                                          4
**  X   X X   O           O                 O   O  O   BAR: W:  0  B: 1
** |X.O.X.X. .O| . . . . .O| .X.X. . .X| .O. .O. .O|  MOVE SCORE:        9.57803
** 24         19 18       13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  44   ROLL: 4-2    BLACK    0- 2 , 10-14
**
**  4   1                                          4
**  X   X X   O           O                 O   O  O
** |X.O.X.X. .O| . . . .X.O| .X. . . .X| .O. .O.X.O|  MOVE SCORE:        5.82806
** 24         19 18       13 12         7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  45   ROLL: 5-1    WHITE   23-18 , 19-18
**
**  4   1                                           4
**  X   X X       O           O                   O   O   O
** |X. .X.X. .O|O. . . .X.O| .X. . . .X| .O. .O.X.O|  MOVE SCORE:       11.07803
** 24          19 18        13 12         7 6          1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  46   ROLL: 6-1    BLACK   14-20 , 22-23
**
**  4                                               4
**  X   X X       O           O                   O   O   O
** |X.X.X.X.X.O|O. . . . .O| .X. . . .X| .O. .O.X.O|  MOVE SCORE:        7.07180
** 24          19 18        13 12         7 6          1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  47   ROLL: 6-3    WHITE   19-13 , 18-15
**
**  4                       1                       4
**  X   X X                 O                   O   O   O
** |X.X.X.X.X. |O. . .O. .O| .X. . . .X| .O. .O.X.O|  MOVE SCORE:        6.53903
** 24          19 18        13 12         7 6          1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  48   ROLL: 2-1    BLACK    2- 4 , 22-23
**
**  4                        1                        4
**  X X   X                  O                  O   O   O
** |X.X.X.X.X. |O. . .O. .O| .X. . . .X| .O.X.O. .O|  MOVE SCORE:        6.37182
** 24         19 18        13 12        7 6          1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  49   ROLL: 4-2    WHITE   15-11 , 13-11
**
**  4                                                 4
**  X X   X                 O   O            O   O   O   BAR: W:  0  B: 1
** |X.X.X.X.X. |O. . . . .O| .O. . . .X| .O.X.O. .O|  MOVE SCORE:        6.50000
** 24         19 18        13 12        7 6          1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  50    ROLL: 5-3    BLACK    NO MOVES
**
**  4                                                 4
**  X X   X                 O   O            O   O   O   BAR: W:  0  B: 1
** |X.X.X.X.X. |O. . . . .O| .O. . . .X| .O.X.O. .O|
** 24         19 18        13 12        7 6          1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  51   ROLL: 6-5    WHITE   13- 7 , 11- 6
**
**  4                                                4
**  X X    X                                 O   O   O   BAR: W:  0  B: 2
** |X.X.X.X.X. |O. . . . .O| .O. . . .O|O.O.X.O. .O|  MOVE SCORE:          7.31561
** 24         19 18       13 12       7 6          1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  52   ROLL: 2-2    BLACK    0- 2 ,  0- 2 ,  4- 6 , 20-22
**
**  4                                                4
**  X X X X                                 O   O X O   BAR: W:  1  B: 0
** |X.X.X.X. . |O. . . . .O| .O. . . .O|X.O. .O.X.O|  MOVE SCORE:          9.74996
** 24         19 18       13 12       7 6          1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  53   ROLL: 5-5    WHITE   25-20 , 11- 6 , 20-15 ,  6- 1
**
**  4                                                5
**  X X X X                                 O   O X O   BAR: W:  0  B: 1
** |X.X.X.X. . |O. . .O. .O| . . . . .O| .O. .O.X.O|  MOVE SCORE:         10.33203
** 24         19 18       13 12       7 6          1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  54   ROLL: 6-3    BLACK    0- 6 , 21-24
**
**  5                                                5
**  X X X                                 0   0 X 0
** |X.X.X.X. . |O. . .O. .O| . . . . .O|X.O. .O.X.O|  MOVE SCORE:        0.06250
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  55   ROLL: 4-2    WHITE    7- 3 , 15-13
**
**  5                                            1   5
**  X X X                 O                 0   0 X 0
** |X.X.X.X. . |O. . . . .O| . . . . . |X.O. .O.X.O|  MOVE SCORE:        5.25000
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  56   ROLL: 3-1    BLACK   21-24 ,  6- 7
**
**  6                                            1   5
**  X X X                 O                 0   0 X 0
** |X.X.X. . . |O. . . . .O| . . . . .X| .O. .O.X.O|  MOVE SCORE:        1.14053
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  57   ROLL: 5-3    WHITE   13-10 , 10- 5
**
**  6                                     1   1   5
**  X X X                                 0   0 X 0
** |X.X.X. . . |O. . . . .O| . . . . .X| .O. .O.X.O|  MOVE SCORE:        6.03903
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  58   ROLL: 2-1    BLACK    2- 4 , 22-23
**
**  6 1                                   1   1   5
**  X X                                   0   0   0
** |X.X.X. . . |O. . . . .O| . . . . .X| .O.X.O.X.O|  MOVE SCORE:        7.55850
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  59   ROLL: 5-2    WHITE   13-11 , 11- 6
**
**  6 1                                   1   1   5
**  X X                                   0   0   0
** |X.X.X. . . |O. . . . . | . . . . .X|O.O.X.O.X.O|  MOVE SCORE:        6.28903
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  60   ROLL: 5-5    BLACK    2- 7 ,  4- 9 ,  9-14 , 14-19
**
**  6 1                                     1   1   5
**  X X                                 X   0   0   0
** |X.X.X. . .X|O. . . . . |. . . . .X|O.O. .O. .O|  MOVE SCORE:       14.50000
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  61   ROLL: 3-2    WHITE    5- 2 ,  6- 4
**
**  6 1                                         1   5
**  X X                                 X   0   0   0
** |X.X.X. . .X|O. . . . . |. . . . .X| .O.O.O.O.O|  MOVE SCORE:        7.12500
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  62   ROLL: 6-3    BLACK    7-10 , 10-16
**
**  6 1                                         1   5
**  X X                                     0   0   0
** |X.X.X. . .X|O. .X. . . |. . . . .X| .O.O.O.O.O|  MOVE SCORE:        4.01953
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  63   ROLL: 6-1    WHITE   18-12 ,  5- 4
**
**  6 1                                         1   5
**  X X                                       0 0   0
** |X.X.X. . .X| . .X. . . |O. . . . .X| .O.O.O.O.O|  MOVE SCORE:        2.19268
** 24         19 18        13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  64   ROLL: 2-1    BLACK   16-17 , 17-19
**
**  6 1                                         1   5
**  X X         X                             0 0   0
** |X.X.X. . .X| . . . . . |O. . . . .X| .O.O.O.O.O|  MOVE SCORE:        9.28900
** 24         19 18        13 12        7 6         1
**
**   STRATEGIES:  W: RUN-OFF   B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  65   ROLL: 6-2    WHITE   12- 6 ,  2- 0
**
**  6 1                                         1   5
**  X X         X                             0 0   0
** |X.X.X. . .X| . . . . . | . . . . .X|O.O.O.O. .O|  MOVE SCORE:        6.00000
** 24         19 18        13 12        7 6         1
**
**   STRATEGIES:  W: BO-OFF    B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  66   ROLL: 5-2    BLACK   19-24 , 19-21
**
**  7 1                                         1   5
**  X X                                       0 0   0
** |X.X.X.X. . | . . . . . | . . . . .X|0.0.0.0. .0|  MOVE SCORE:        8.06250
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: BO-OFF    B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  67   ROLL: 3-3    WHITE    3- 0 ,  3- 0 ,  4- 1 ,  3- 0
**
**  7 1                                           6
**  X X                                           0
** |X.X.X.X. . | . . . . . | . . . . .X|0.0.0. . .0|  MOVE SCORE:       14.00000
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: BO-OFF    B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  68   ROLL: 6-1    BLACK    7-13 , 21-22
**
**  7 1                                           6
**  X X X                                         0
** |X.X.X. . . | . . . . .X| . . . . . |0.0.0. . .0|  MOVE SCORE:        2.75000
** 24         19 18       13 12        7 6         1
**
**   STRATEGIES:  W: BO-OFF    B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  69   ROLL: 5-5    WHITE    6- 1 ,  5- 0 ,  4- 0 ,  1- 0
**
**  7 1                                                 6
**  X X X                                               0
** |X.X.X. . . |. . . . .X|. . . . .|. . . . .0|  MOVE SCORE:      14.00000
** 24        19 18       13 12       7 6       1
**
**   STRATEGIES:  W: BO-OFF    B: RUN-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  70   ROLL: 6-5    BLACK   13-19 , 22-25
**
**  7 1                                                 6
**  X X                                                 0
** |X.X.X. . .X|. . . . . |. . . . .|. . . . .0|  MOVE SCORE:       6.00000
** 24        19 18       13 12       7 6       1
**
**   STRATEGIES:  W: BO-OFF    B: BO-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  71   ROLL: 6-5    WHITE    1- 0 ,  1- 0
**
**  7 1                                                 4
**  X X                                                 0
** |X.X.X. . .X|. . . . . |. . . . .|. . . . .0|  MOVE SCORE:       8.00000
** 24        19 18       13 12       7 6       1
**
**   STRATEGIES:  W: BO-OFF    B: BO-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  72   ROLL: 6-4    BLACK   19-25 , 22-25
**
**  7 1                                                 4
**  X X                                                 0
** |X.X. . . . |. . . . . |. . . . . |. . . . . .0|  MOVE SCORE:        8.00000
** 24         19 18       13 12       7 6         1
**
**   STRATEGIES:  W: BO-OFF    B: BO-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  73   ROLL: 6-4    WHITE    1- 0 ,  1- 0
**
**  7 1                                                 2
**  X X                                                 0
** |X.X. . . . |. . . . . |. . . . . |. . . . . .0|  MOVE SCORE:        8.00000
** 24         19 18       13 12       7 6         1
**
**   STRATEGIES:  W: BO-OFF    B: BO-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  74   ROLL: 2-1    BLACK   23-25 , 24-25
**
**  6                                                   2
**  X X                                                 0
** |X.X. . . . |. . . . . |. . . . . |. . . . . .0|  MOVE SCORE:        8.00000
** 24         19 18       13 12       7 6         1
**
**   STRATEGIES:  W: BO-OFF    B: BO-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  75   ROLL: 6-2    WHITE    1- 0 ,  1- 0
**
**  6
**  X X                                                 O
** |X.X. . . . .|. . . . .|. . . . .|. . . . .O|  MOVE SCORE:        8.00000
** 24         19 18       13 12       7 6        1
**
**   STRATEGIES:  W: BO-OFF    B: BO-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  76   ROLL: 4-3    BLACK   23-25 , 23-25
**
**  6
**  X                                                   O
** |X. . . . .|. . . . .|. . . . .|. . . . .O|  MOVE SCORE:        8.00000
** 24         19 18       13 12       7 6        1
**
**   STRATEGIES:  W: BO-OFF    B: BO-OFF
**
*******************
```

```
*******************
**
**   MOVE NO:  77   ROLL: 4-1    WHITE    1- 0 ,  1- 0
**
**  6
**  X
** |X. . . . .|. . . . .|. . . . .|. . . . . |  MOVE SCORE:        8.00000
** 24         19 18       13 12       7 6        1
**
**   STRATEGIES:  W: BO-OFF    B: BO-OFF
**
*******************
```

```
/*****************************************************************/
/*                                                               */
/*                    M O D U L E   G A M M O N                  */
/*                                                               */
/*****************************************************************/

 GAMMON:  PROC OPTIONS(MAIN);
 /* GAMX ACTS AS AN INTERFACE BETWEEN THE TERMINAL USER AND THE
    REST OF THE PROGRAM. IT ACCEPTS TWO-LETTER COMMANDS AND THEN
    CALLS ADMIN AT THE APPROPRIATE ENTRY POINT.    */
DCL CODE CHAR(2),NEWDEL FIXED BIN(31,15);
ON CONVERSION GO TO WHAT;
DCL ADM1D ENTRY(FIXED BIN(31,15)) EXTERNAL,
ADM1S ENTRY(FIXED BIN,FIXED BIN) EXTERNAL;
PUT SKIP EDIT('WELCOME GAME LOVERS')(COL(1),A);
TOP1: CALL ADMIN1;
TOP2: PUT SKIP(2) LIST('*** ? ***'); GET EDIT(CODE) (COL(1),A(2));
TOP3:  IF CODE = 'GO' THEN DO; CALL ADM1$; GO TO TOP2; END;
         ELSE IF CODE = 'XX' THEN DO;
         PUT SKIP(2) LIST('THANKS FOR THE GAME.  NO HARD FEELINGS, OK?');
         STOP; END;
         ELSE IF CODE = 'RE' THEN GO TO TOP1;
         ELSE IF CODE  = 'GR' THEN CALL ADM1G;
         ELSE IF CODE = 'TR' THEN CALL ADM1T;
         ELSE IF CODE = 'CS' THEN DO;
           PUT SKIP LIST('WHOSE STRATEGY DO YOU WISH TO CHANGE?(1=WHT,2=BLK)');
           GET LIST(J);
           PUT SKIP LIST('ENTER CODE FOR NEW STRATEGY');
           GET LIST(K);
           CALL ADM1S(J,K);  GO TO TOP2; END;
         ELSE IF CODE = 'CD' THEN DO;
            PUT SKIP LIST('ENTER NEW VALUE FOR DEL');
           GET LIST(NEWDEL);
           CALL ADM1D(NEWDEL); GO TO TOP2; END;
         ELSE IF CODE = 'EV' THEN CALL ADM1E;
         ELSE IF CODE = 'GE' THEN CALL ADMGE;
         ELSE IF CODE = 'DI' THEN CALL ADMDI;
         ELSE IF CODE = 'SO' THEN CALL ADMSO;
         ELSE IF CODE = 'LO' THEN CALL ADMLO;
         ELSE IF CODE = 'LI' THEN CALL ADMLI;
         ELSE IF CODE = 'SW' THEN CALL ADMSW;
         ELSE IF CODE = 'ST' THEN CALL ADMST;
         ELSE GO TO WHAT;
         GO TO TOP2;
 WHAT:PUT SKIP LIST('I DO NOT RECOGNIZE THAT COMMAND. TRY AGAIN.');
       GO TO TOP2;
 END GAMMON;
```

```
/*****************************************************************/
/*                                                               */
/*                    M O D U L E   A D M I N                    */
/*                                                               */
/*****************************************************************/

ADMIN1: PROC;
DCL     (BCODE,
         SCODEW,SCODEB,CCODEB,CCODEW,
         ACTIVE,PASSIVE,JENT,KENT,WINNER,WINS(2),TALLY(2),GAMECTR,FIRSTMOVER,
         MVNO,D1,D2,A,LVL,DD1,DD2,KEEPLVL,STEST,DBLCTR,GENLIMIT,KEEPCOUNT,
         (WB,GB)(24),EVALBOARD(24),MODE,
         STAT(2,0:12),SAVESTAT(2,0:12),
         STRAT(2,10,3,10),
         CS(2), S1,
         OPN(27,5),
         LIST(50,4) , LISTCNT,
         MOVES(2,4),
         IMPLIST(4)) FIXED BIN(15) STATIC ,
         PLAYER FIXED BIN(15),
         SEED FIXED BIN(31) ,
         (PE(0:6),PH(6,0:5)) FIXED BIN(15,10) STATIC,
         (C(2,10,3,10), SCR,DEL INITIAL(4),
         X(50),
         KEEPX(3,10)) FIXED BIN(31,15) STATIC,
         NEWDEL FIXED BIN(31,15),
         (ENTER,DOUBLES, BEAROFF,TRACE, REDOX,TIE,SECOND_PAIR,RISKFLAG,
          BATCH,BLKEVAL,KEEPTRACE,SQUEEZE,GAMFLAG,CONTINUE,
         (FROM,TO)(4,24),
         BLOT(2,24)) BIT(1) STATIC,
         (T,F,
         (TO_STR,FROM_STR)(2)) BIT(24) STATIC,
         BUFF CHAR(80) VAR, YN CHAR(1),
         1 TMOVE STATIC,2 R, 3 (B1,B2,B3,B4,B41,B5,B6) BIT(24),
                 2 BO, 3 (B7,B8,B9,B10,B11,B12,B13,B14) BIT(6),
                 2 ED, 3 (B15,B16) BIT(2);
DCL (RET1,RET2) LABEL;
DCL ROLL ENTRY(FIXED BIN(31),FIXED BIN(15),FIXED BIN(15));
DCL MVEVAL ENTRY(FIXED BIN(15),,FIXED BIN(15),,,,,FIXED BIN(15),,,,,,,,,);
DCL MVLIST ENTRY(FIXED BIN(15),,FIXED BIN(15),FIXED BIN(15),,,,);
DCL GETSTRT RETURNS(FIXED BIN(15));
ON CONVERSION BEGIN;PUT SKIP LIST('CONVERSION ERROR');GO TO INITIAL; END;
INITIAL:
  STEST = 0;
  PUT SKIP LIST('STANDARD TEST GAME? 1 OR 0');
  GET LIST(J);
  IF J = 1 THEN DO;
```

```
      BLKEVAL,BATCH,TRACE,GAMFLAG,CONTINUE = '0'B;
      STEST, BCODE,SCODEW,SCODEB,ACTIVE,MODE,FIRSTMOVER = 1;
      CCODEW,CCODEB,GAMECTR = 0;
      PASSIVE = 2;
      SEED = 235;
      GO TO TOP1;  END;
   PUT SKIP LIST(' IS THIS A BATCH GAME? 1 OR 0');
   GET EDIT(J) (COL(1),F(1));
   IF J = 1 THEN DO; BATCH = '1'B;
   PUT SKIP LIST('ENTER MODE'); GET LIST (MODE);
   PUT SKIP LIST('OUTPUT TO FILE? 1 OR 0');
   GET LIST(J);
   IF J = 1 THEN GAMFLAG = '1'B; ELSE GAMFLAG = '0'B; END;
   ELSE DO;  GAMFLAG,BATCH = '0'B; MODE = 1;  END;
   PUT SKIP LIST('ENTER CODE FOR BOARD SET UP(1=STANDARD)') ;
   GET  LIST(BCODE);
   PUT SKIP LIST('ENTER CODES FOR WHITE AND BLACK STRATEGY SETS');
   GET LIST(SCODEW,SCODEB);
   PUT SKIP LIST('ENTER CODES FOR WHITE AND BLACK COEFFICIENT SETS');
   GET LIST(CCODEW,CCODEB);
   PUT SKIP LIST('WHO HAS FIRST MOVE?(1=WHT, 2=BLK)');
   GET  LIST(ACTIVE);
   FIRSTMOVER = ACTIVE;
   IF ACTIVE = 1 THEN PASSIVE = 2; ELSE PASSIVE = 1;
   PUT SKIP LIST('DO YOU WISH YOUR MOVES EVALUATED?(1 OR 0)');
   GET EDIT(J)(COL(1),F(1));
   IF J = 1 THEN BLKEVAL = '1'B; ELSE BLKEVAL = '0'B;
   PUT SKIP LIST('ENTER AN ODD INTEGER TO INITIALIZE THE DICE ROLLER');
   GET  LIST(SEED);
   CALL ROLL(SEED,D1,D2);
   PUT SKIP LIST('TRACE? 1 OR 0') ;
   GET EDIT(J) (COL(1),F(1));
   IF J = 1 THEN TRACE = '1'B;
      ELSE TRACE = '0'B;
   REVERT CONVERSION;
   GAMECTR = 0; WINS,TALLY = 0; CONTINUE = '0'B;
   GO TO TOP1;
TOP:
  CONTINUE  = '1'B;
 TOP1: IMPLIST = 0;
 REDOX = '0'B;
 RISKFLAG = '1'B;
 GENLIMIT = 30;
 SEED = 0;
 SCR = 0;
 X = 0;
 MVNO = 0; MOVES = 0;
 CALL SETUP(BATCH,BCODE,SCODEW,SCODEB,CCODEW,CCODEB,ACTIVE,
```

```
IMPLIST,WB,GB,STAT,STRAT,C,CS,PE,PH,OPN,BLOT,FROM_STR,TO_STR,TRACE,CONTINUE);
  KEEPLVL,DBLCTR = 1;
  A = 2;
  IF STEST = 0 THEN CALL PUT1
       (MVNO,ACTIVE,GB,MOVES,CS,STAT,D1,D2,SCR,BATCH,GAMFLAG);
  MOVES = 0;
  MVNO = 1;
  GO TO GAME_TOP;
 ADM1$:
   ENTRY;
 GAME_TOP:
   IF TRACE THEN PUT SKIP LIST('ACTIVE PLAYER:'||ACTIVE);
   IF MODE = 1 | MODE = 2 | FIRSTMOVER = ACTIVE THEN CALL ROLL(0,D1,D2);
   IF ¬BATCH THEN DO;
      IF ACTIVE = 2 THEN PUT SKIP(2) LIST('BLACK DICE ROLL:'||D1||D2);
         ELSE   PUT SKIP(2) LIST('WHITE DICE ROLL:'||D1||D2); END;
   IF D1 = D2 THEN DO; DOUBLES = '1'B; RISKFLAG = '0'B; END;
      ELSE DOUBLES = '0'B;
 DBL_LOOP:
   T = TO_STR(ACTIVE);
   F = FROM_STR(ACTIVE);
   IF ACTIVE = 1 THEN WB = GB;
       ELSE DO; DO I = 1 TO 24; WB(I) = -GB(25-I); END; END;
   IF MVNO = 1 THEN IF BATCH THEN IF BCODE = 1 THEN DO;
      J1 = 10*D1 + D2;
      DO I = 1 TO 27;
         IF J1 = OPN(I,1) THEN DO;
            IF DOUBLES THEN IF SECOND_PAIR THEN I = I + 1;
            DO J2 = 2 TO 5;
            MOVES(DBLCTR,J2-1) = OPN(I,J2); END;
            IMPLIST = MOVES(DBLCTR,*);
            GO TO IMPLEMENT;
        END;
     END;
  END;
   IF TRACE THEN PUT SKIP LIST('CALLING MASK,T= '||T||'  F= '||F);
  CALL MASK(D1,D2,F,T,FROM,TO,TRACE);
  IF ¬BATCH THEN IF ACTIVE = 2 THEN GO TO BLACK_MOVE;
   CS(ACTIVE) = GETSTRT;
   IF TRACE THEN PUT SKIP LIST(' STRATEGY SELECTION:'||CS(ACTIVE));
   IF ENTER THEN GO TO DO_ENTER;
   IF SQUEEZE THEN GO TO DO_SQUEEZE;
 GEN_LOOP:
  S1 = CS(ACTIVE);
   RET1 = LOAD_MOVES;RET2 = NO_MOVES; GO TO CYCLE;
 LOAD_MOVES:
  IF DOUBLES THEN IF SECOND_PAIR THEN DBLCTR = 2;
  MOVES(DBLCTR,*) = LIST(1,*);
```

```
 IMPLIST = LIST(1,*);
IMPLEMENT:
  IF ACTIVE = 1 | BATCH  THEN SCR = SCR + X(1);
  IF ¬BATCH THEN IF BLKEVAL THEN DO;
     LIST(1,*) = IMPLIST;
      DO I = 1 TO 3;
        CALL MVEVAL(2,CS(2),I,STRAT,STAT,WB,LIST,1,X,KEEPX,C,PE,PH,RISKFLAG,
              REDOX,TRACE,ENTER);END;
     SCR = SCR + X(1);END;
 IF ¬BATCH THEN IF ACTIVE = 1 | BLKEVAL THEN DO;
     EVALBOARD = WB;LIST(1,*) = IMPLIST; SAVESTAT = STAT; END;
 IF TRACE THEN PUT SKIP LIST('CALLING MVIMP,DBLCTR =',DBLCTR);
 CALL MVIMP(ACTIVE,D1,D2,STAT,WB,GB,IMPLIST,SCR,BLOT,BEAROFF,
      TO_STR,FROM_STR,TRACE);
 IF DOUBLES THEN IF DBLCTR = 1 THEN DO;
     RISKFLAG = '1'B;
     SECOND_PAIR = '1'B; DBLCTR = 2;
     IF ACTIVE = 2 THEN IF BATCH = '0'B THEN DO;
     IF STAT(2,3) <= 0 THEN GO TO END_GAME;
     CALL CHECK;
           DO I = 1 TO 24; WB(I) = -GB(25-I); END;
           IMPLIST = MOVES(2,*); GO TO IMPLEMENT; END;
     A = 2;
     GO TO DBL_LOOP;  END;
 IF TRACE THEN PUT SKIP LIST('CALLING PUT1');
 PUTBD:
  CALL PUT1(MVNO,ACTIVE,GB,MOVES,CS,STAT,D1,D2,SCR,BATCH,GAMFLAG);
 GO TO BACK;
NO_MOVES:
  PUT SKIP LIST('NO MOVES GENERATED');
 GO TO PUTBD;
BLACK_MOVE:
 PUT SKIP(2) LIST('****  ENTER MOVES, PLEASE  ****') ;
 IF DOUBLES THEN I1 = 2;ELSE I1 = 1;
  GET  LIST(((MOVES(I,J) DO J = 1 TO 4) DO I = 1 TO I1));
 DO I = 1 TO I1; DO J = 1 TO 4;
 MOVES(I,J) = 25 - MOVES(I,J); END; END;
 IMPLIST = MOVES(1,*);
 IF STAT(2,9) > 0 THEN ENTER = '1'B;ELSE ENTER = '0'B;
 IF STAT(2,2) <= 0 THEN BEAROFF = '1'B;
   ELSE BEAROFF = '0'B;
  CALL CHECK;
 GO TO IMPLEMENT;
BACK:
  IF STAT(ACTIVE,3) <= 0 THEN GO TO END_GAME;
 PASSIVE = ACTIVE;
 IF ACTIVE = 1 THEN ACTIVE = 2;
      ELSE ACTIVE = 1;
```

```
    MOVES = 0;
    SCR = 0;
    DBLCTR = 1;
    SECOND_PAIR = '0'B;
    RISKFLAG  = '1'B;
    MVNO = MVNO + 1;
    A = 2;
    IF BATCH THEN GO TO GAME_TOP;
     RETURN;
   END_GAME:
    PUT SKIP(2) LIST('*****  E N D   O F   G A M E   *****');
    IF ACTIVE = 1 THEN PUT SKIP LIST('WHITE WINS');
         ELSE PUT SKIP LIST('BLACK WINS');
    K = 167 - STAT(PASSIVE,1);
    PUT SKIP LIST('POINT VALUE OF WIN = ',K);
     IF MODE = 1 | MODE = 2 THEN RETURN;
     GAMECTR = GAMECTR + 1;
     WINS(ACTIVE) = WINS(ACTIVE) + 1;
     TALLY(ACTIVE) = TALLY(ACTIVE) + K;
     IF MODE = 3 THEN IF GAMECTR = 2 THEN GO TO FINIS;
     IF MODE = 4 THEN IF GAMECTR = 10 THEN GO TO FINIS;
     IF FIRSTMOVER = 1 THEN FIRSTMOVER = 2;
        ELSE FIRSTMOVER = 1;
     ACTIVE = FIRSTMOVER;
     GO TO TOP;
  FINIS:
     PUT SKIP(5) LIST('****  END OF SERIES  ****');
     PUT SKIP(2) LIST('SUMMARY');
     PUT SKIP LIST('      WHITE:  WINS='||WINS(1)||'  POINTS='||TALLY(1));
     PUT SKIP LIST('      BLACK:  WINS='||WINS(2)||'  POINTS='||TALLY(2));
     K = WINS(1) - WINS(2);
     IF K > 0 THEN PUT SKIP(2) LIST('WHITE IS GRAND WINNER');
        ELSE IF K < 0 THEN PUT SKIP(2) LIST('BLACK IS GRAND WINNER');
        ELSE PUT SKIP LIST('SERIES IS TIED');
     K = ABS(K);
     PUT SKIP(2) LIST('NET SCORE = '||K);
     RETURN;
   DO_ENTER:
    IF TRACE THEN PUT SKIP LIST('IN DO_ENTER BLOCK OF ADMIN1');
    IF STAT(PASSIVE,6) >= 6 THEN GO TO NO_MOVES;
    K2 = STAT(ACTIVE,9);  /* NO OF MEN TO ENTER  */
    IF K2 >= 2 THEN A = 2;  ELSE A = 1;
     RET1 = ENTER1;  RET2= NO_MOVES;  GO TO CYCLE;
    ENTER1: IF A = 2 THEN GO TO A2;
    MOVES(DBLCTR,*) = LIST(1,*);
    J = 25 - LIST(1,2);
      DD1 = D1; DD2 = D2;
     IF J = D1 THEN DO;
```

```
   IF DOUBLES THEN DO; D2 = 0; SUBSTR(F,D1,1) = '1'B; GO TO ENTER3;END;
   D1 = 0; SUBSTR(F,DD1,1) = '1'B;  END;
   ELSE DO;
   D2  = 0; SUBSTR(F,DD2,1) = '1'B; END;
   ENTER3: ENTER = '0'B;
   CALL MASK(D1,D2,F,T,FROM,TO,TRACE);
   JENT = WB(LIST(1,2));
   IF JENT >= 0 THEN KENT = JENT + 1;
   ELSE IF JENT = -1 THEN KENT = 1;
   ELSE PUT SKIP LIST('INVALID ENTER MOVE DETECTED IN ADMIN');
   WB(LIST(1,2)) = KENT;
   STAT(ACTIVE,9) = STAT(ACTIVE,9) - 1;
   RET1 = ENTER2; RET2 = OUT1; GO TO CYCLE;
ENTER2:
   MOVES(DBLCTR,3) = LIST(1,1);
  MOVES(DBLCTR,4) = LIST(1,2);
OUT1:  /*  RESTORE WB AND STAT 9  */
   WB(MOVES(DBLCTR,2)) = JENT;
   STAT(ACTIVE,9) = STAT(ACTIVE,9) + 1;
   D1 = DD1; D2 = DD2;
  GO TO OUT;
 A2:  MOVES(DBLCTR,*) = LIST(1,*);
 OUT: IF DOUBLES THEN IF SECOND_PAIR THEN DBLCTR = 2;
  IMPLIST = MOVES(DBLCTR,*);
  GO TO IMPLEMENT;
DO_SQUEEZE:
   IF TRACE THEN PUT SKIP LIST('IN SQUEEZE BLOCK OF ADMIN1');
   A = 1;  DD1 = D1;  DD2 = D2;
   J = STAT(ACTIVE,7);
  /*  SQUEEZE IN WITH D1?  */
   IF WB(J-D1) > -1 THEN DO;
   MOVES(DBLCTR,1) = J;
   MOVES(DBLCTR,2) = J - D1;
   D1 = 0;
   GO TO SQ5;   END;
   /* ELSE CAN WE SQUEEZE IN WITH D2   */
   ELSE IF ¬DOUBLES THEN IF J - D2 < 7 THEN IF WB(J-D2) > -1 THEN DO;
      MOVES(DBLCTR,1) = J;
      MOVES(DBLCTR,2) = J - D2;
      D2 = 0;  GO TO SQ5;  END;
   /* CANT SQUEEZE   */
   ELSE GO TO GEN_LOOP;
SQ5:
   BEAROFF = '1'B;
   IF DOUBLES THEN IF D1 = 0 THEN DO; D1 = DD1; D2 = 0;  END;
   J = MOVES(DBLCTR,1);
   K = MOVES(DBLCTR,2);
   KENT = WB(K);
```

```
   IF KENT = -1 THEN KENT = 1;
   ELSE WB(K) = WB(K) + 1;
   WB(J) = WB(J) - 1;
   IF WB(J) = 0 THEN SUBSTR(F,J,1) = '0'B;
   SUBSTR(T,K,1) = '1'B;
   CALL MASK(D1,D2,F,T,FROM,TO,TRACE);
   CS(ACTIVE) = 6;
  S1 = 6;
   RET1 = SQ6; RET2 = RET; GO TO CYCLE;
SQ6:
   MOVES(DBLCTR,3) = LIST(1,1);
   MOVES(DBLCTR,4) = LIST(1,2);
 RET:
   IMPLIST = MOVES(DBLCTR,*);
   WB(K) = KENT;  WB(J) = WB(J) + 1;
   D1 = DD1; D2 = DD2;
   GO TO IMPLEMENT;
CYCLE:
   LISTCNT = 0;
   KEEPTRACE = TRACE;
   IF TRACE THEN IF ENTER | BEAROFF | MOD(MVNO,10) = 0 THEN GO TO CYC1;
   ELSE TRACE = '0'B;
CYC1:
   DO LVL = 1 TO 3 WHILE(LISTCNT <= GENLIMIT);
      LISTCNT = 0;
      IF TRACE THEN PUT SKIP LIST('CALLING MVGEN,LEVEL',LVL);
      CALL MVGEN(ACTIVE,S1,LVL,A,D1,D2,WB,TMOVE,STRAT,STAT,BLOT,FROM,
       TO,ENTER,TRACE);
      IF TRACE THEN PUT SKIP LIST('CALLING MVLIST');
      CALL MVLIST(A,DOUBLES,D1,D2,TMOVE,LIST,LISTCNT,TRACE);
      IF BEAROFF THEN GO TO CYC2;
      END;
CYC2:
   IF LISTCNT > 0  THEN GO TO CYC3;
   IF ENTER THEN GO TO CYC4;
   CALL MVGEN(ACTIVE,99,LVL,A,D1,D2,WB,TMOVE,STRAT,STAT,BLOT,FROM,TO,
          ENTER,TRACE);
   CALL MVLIST(A,DOUBLES,D1,D2,TMOVE,LIST,LISTCNT,TRACE);
   IF LISTCNT > 0 THEN GO TO CYC3;
CYC4:
   CALL MVLIST(1,DOUBLES,D1,0,TMOVE,LIST,LISTCNT,TRACE);
   IF  LISTCNT > 0 THEN GO TO CYC3;
      ELSE IF DOUBLES THEN GO TO CYC21;
   CALL MVLIST(1,DOUBLES,0,D2,TMOVE,LIST,LISTCNT,TRACE);
   IF LISTCNT > 0 THEN GO TO CYC3;
CYC21:
   KEEPTRACE = TRACE;
   GO TO RET2;
```

```
CYC3:
   TIE = '1'B; KEEPCOUNT = LISTCNT;
   DO LVL = 1 TO 3 WHILE (TIE);
      IF TRACE THEN PUT SKIP LIST('CALL MVEVAL, LEVEL'||LVL);
      CALL MVEVAL(ACTIVE,S1,LVL,STRAT,STAT,WB,LIST,LISTCNT,X,
               KEEPX,C,PE,PH,RISKFLAG,REDOX,TRACE,ENTER);
      KEEPLVL = LVL;
      IF TRACE THEN PUT SKIP LIST('CALLING MVPICK');
      CALL MVPICK(LIST,LISTCNT,X,TIE,DEL,TRACE);
   END;
   TRACE = KEEPTRACE;
   GO TO RET1;
ADM1G:  ENTRY;
   DCL (TACT,GRADE) FIXED BIN(15), (Y2,Y3,Y4) FIXED BIN(31,15);
   DCL OLDC FIXED BIN(31,15);
   DCL (X1,X2,X3,X4,X5,XDELTA,XPOS,XNEG) FLOAT DECIMAL;
   ON FIXEDOVERFLOW BEGIN;
   PUT SKIP DATA(TACT,OLDC,X2,X3,XDELTA);  GO TO GRET; END;
   PUT SKIP LIST('ENTER GRADE');  GET LIST(GRADE);
GRADETOP:
   I1 = PASSIVE; I2 = CS(PASSIVE);
  KEEPTRACE = '0'B;
   REDOX = '1'B;   XSCR,XPOS,XNEG  = 0;   LISTCNT = 1;
   IF TRACE THEN DO;  KEEPTRACE = TRACE; TRACE = '0'B; END;
   DO I = 1 TO KEEPLVL;
      CALL MVEVAL(I1,I2,I,STRAT,SAVESTAT,EVALBOARD,LIST,LISTCNT,
         X,KEEPX,C,PE,PH,RISKFLAG,REDOX,TRACE,ENTER);END;
   XSCR = XSCR + X(1);
   XPOS = XPOS + X(2);
   XNEG = XNEG + X(3);
   IF KEEPTRACE THEN TRACE = KEEPTRACE;
   IF EVALFLAG THEN DO;
      PUT SKIP(2) LIST('MOVE SCORE = '||XSCR);
      PUT SKIP(2) LIST('ENTER GRADE(0 TO QUIT)');
      GET LIST(GRADE);
      IF GRADE = 0 THEN GO TO GRET;   END;
   XPOS = X(2);
   XNEG = X(3);
   X5  = GRADE;
   DO I3 = 1 TO KEEPLVL;
      DO I4 = 1 TO 10;
         TACT = STRAT(I1,I2,I3,I4);
         IF TACT = 0 THEN GO TO I3_END;
         Y2 = KEEPX(I3,I4);
         X2 = Y2;
         IF X2 = 0 THEN GO TO I4_END;
         OLDC = C(I1,I2,I3,I4);
         X1 = OLDC;
```

```
          IF X1 = 0 THEN X1 = 1;
          IF X2 > 0 THEN DO;
             IF XPOS ¬= 0 THEN X3 = X2/XPOS;
                 ELSE X3 = 1;  END;
             ELSE DO;
                IF XNEG ¬= 0 THEN X3 = X2/XNEG;
                ELSE X3 = 1;    END;
          XDELTA = ABS((X1/4)*(X5/5)*X3);
          IF X1 = 0 THEN DO;  IF XDELTA < 0 THEN X4 = 0;
                 ELSE X4 = XDELTA;  GO TO REPLACE;  END;
          IF X5*X2 < 0 THEN X4 = X1 - XDELTA;
             ELSE X4 = X1 + XDELTA;
          REPLACE:  C(I1,I2,I3,I4) = X4;
          IF TRACE THEN DO;
             Y3 = XDELTA;  Y4 = X4;
PUT SKIP LIST('TACTIC'||TACT||'  OLD C = '||OLDC||'  PART.SCR.='||Y2
   ||'  DELTA ='||Y3||'  NEW C ='||Y4);  END;
      I4_END: END;
   I3_END: END;
 GRET:
   EVALFLAG,REDOX = '0'B; RETURN;
ADM1S:
   ENTRY(PLAYER,NEW_S); CS(PLAYER) = NEW_S; PUT SKIP DATA(CS); RETURN;
ADM1T:
   ENTRY; TRACE = ¬TRACE; PUT SKIP DATA(TRACE); RETURN;
ADMGE:
  ENTRY; RETURN;
ADM1E:
   ENTRY;
   DCL EVALFLAG BIT(1) INIT('0'B) STATIC;
   PUT SKIP LIST('ENTER MOVE TO BE EVALUATED');
   GET LIST ((LIST(1,I) DO I = 1 TO 4));
   EVALFLAG = '1'B;
   IF PASSIVE = 2 THEN LIST(1,*) = 25 - LIST(1,*);
   KEEPLVL = 3;
   GO TO GRADETOP;
   RETURN;
ADM1D:
    ENTRY(NEWDEL); DEL = NEWDEL; RETURN;
    /*****************************************************************/
    /*                    SUBROUTINE CHECK                           */
    /*****************************************************************/
CHECK:   PROC;
   /* CHECK DETERMINES THE LEGALITY OF A MOVE ENTERED AT THE
      TERMINAL */
   DCL(P(4),SAVEJ) FIXED BIN(15),
   MV = DBLCTR;
   P = MOVES(MV,*);
```

```
   IF ENTER THEN GO TO CHK_ENT;
   IF BEAROFF THEN GO TO CHK_BEAR;
   IF P(2) = 25 THEN GO TO M1;
   IF P(4) = 25 THEN GO TO M2;
C1:
   J1=P(1)-P(2);
   J2=P(3)-P(4);
   IF J1=D1 THEN IF J2 ¬= D2 THEN GO TO M3;
                                ELSE GO TO C2;
   IF J2=D1 THEN IF J1 ¬= D2 THEN GO TO M4;
                                ELSE GO TO C2;
   GO TO M41;
C2:
  DO J = 1,3;
      IF J = 3 THEN IF P(2) = P(3) THEN GO TO C21;
      IF ENTER THEN GO TO C21;
      IF SUBSTR(F,25-P(J),1) = '0'B THEN DO;
         SAVEJ=J; GO TO M5; END;
   C21: IF SUBSTR(T,25-P(J+1),1)='0'B THEN DO;
         SAVEJ=J; GO TO M6; END;
   END;
   RETURN;
CHK_ENT:
   J=STAT(ACTIVE,9); /* NO. OF MEN ON BAR */
   IF J >  2 THEN J = 2;
   K=0;
C3:
   DO I=1,3;
      IF P(I)=25 THEN K=K+1;END;
      IF K < J THEN IF P(1) ¬= 25 | P(3) ¬= 25 THEN GO TO M7;
      GO TO C1;
CHK_BEAR:
   IF P(2)=0 THEN IF  P(1)>D1 THEN GO TO M8;
   IF P(4)=0 THEN IF  P(3) > D1 THEN GO  TO M9;
   RETURN;
M1:PUT SKIP LIST('ARE YOU SURE YOU HAVE NO MOVES?');GO TO GET;
M2:PUT SKIP LIST('THIS IS ONLY A PARTIAL MOVE');GO TO GET;
M3:PUT SKIP LIST('SECOND PART OF MOVE'||MV||'NOT = D2');GO TO GET;
M4:PUT SKIP LIST('FIRST PART OF MOVE'||MV||'NOT = D1');GO TO GET;
M41:PUT SKIP LIST('INCREMENTS OF MV'||MV||' DO NOT MATCH DICE');GO TO GET;
M5:PUT SKIP LIST('IN MOVE'||MV||' MOVING FROM ILLEGAL POSITION:'||25 -
   P(SAVEJ)); GO TO GET;
M6:PUT SKIP LIST('IN MOVE'||MV||' MOVING TO ILLEGAL POSITION:'||
   25 - P(SAVEJ)); GO TO GET;
M7:PUT SKIP LIST('MOVE DOES NOT ENTER ENOUGH MEN');GO TO GET;
M8:M9:PUT SKIP LIST('BEAROFF ILLEGAL');
GET:
   PUT SKIP LIST('...ENTER 1 TO CONTINUE, 0 TO RE-ENTER MOVES');
```

```
   GET EDIT(K)(COL(1),F(1));
   IF K = 0 THEN DO;
      IF DOUBLES THEN PUT SKIP LIST('***RE-ENTER MOVE'||MV);
        ELSE PUT SKIP LIST('*** RE-ENTER MOVE');
      GET LIST((MOVES(MV,I) DO I = 1 TO 4));
      MOVES(MV,*) = 25 - MOVES(MV,*);
   IMPLIST = MOVES(MV,*);
   END;
   RETURN;
END CHECK;
    /*****************************************************************/
    /*                    SUBROUTINE GETSTRT                         */
    /*****************************************************************/
GETSTRT: PROC RETURNS(FIXED BIN(15));
DCL (
   S,
   OLD_STRAT,
   OLAP,
   RPA,
   IHOLD,
   HEHOLD,
   BP,
   ((VRUNOLP INIT(20),
   VRUNRPA INIT(15),
   VPOSRPA INIT(20),
   VBLKRPA INIT(15),
   VBACKRPA INIT(-50)) STATIC)) FIXED BIN(15);
   SQUEEZE,  ENTER = '0'B;
   OLD_STRAT = CS(ACTIVE);
  /*  CHECK FOR ENTER CONDITIONS   */
   IF STAT(ACTIVE,9) > 0 THEN DO;
      ENTER = '1'B;
     IF BEAROFF THEN DO;
      BEAROFF = '0'B;
      RETURN(2); END;
      GO TO S4;
   END;
  /*  CHECK FOR SQUEEZE CONDITIONS   */
   J = STAT(ACTIVE,2);
   IF J > 0 THEN IF J <= D1 THEN DO;
      K1 = STAT(ACTIVE,7);
      IF WB(K1) > 1 | STAT(ACTIVE,8) > 6 THEN GO TO S4;
      SQUEEZE = '1'B;
      RETURN(OLD_STRAT);   END;
  /*  BEARING OFF CONDITIONS   */
   S3:  IF J <= 0 THEN DO;
   IF STAT(ACTIVE,3) = 1 THEN A = 1;
   BEAROFF = '1'B;
```

```
    IF STAT(ACTIVE,4) >= 0 | STAT(PASSIVE,9) >0
    THEN RETURN (6);
    ELSE RETURN (7);  END;
   S4: S = CS(PASSIVE);
    OLAP = STAT(1,4);
    /*  OPPONENT IS BEARING OFF    */
    IF S = 6 | S = 7 | STAT(PASSIVE,2) <= 0  THEN DO;
       IF OLAP >= VRUNOLP | STAT(ACTIVE,9) > 0 THEN RETURN(3);
       ELSE RETURN(2);  END;
   /*  OPPONENT IS PLAYING A BACK GAME  */
    IF S = 5 THEN RETURN(3);
    /*  OPPONENT IS BLOCKING   */
    RPA = STAT(ACTIVE,1) - STAT(PASSIVE,1);
    IF S = 4 THEN DO;
       IF RPA >= VRUNRPA THEN RETURN(3);
       IF ABS(RPA) <= VPOSRPA THEN RETURN(1);
    END;
    /*  OPONENT IF RUNNING OR PLAYING POSITION   */
    IF STAT(ACTIVE,5) > 3
       |  STAT(PASSIVE,9) > 2
        |  (STAT(PASSIVE,9) > 0 & STAT(ACTIVE,6) > 3)
       | (RPA >= VRUNRPA & OLAP <= VRUNOLP)
    THEN RETURN(2);
    IF RPA <= VBLKRPA THEN DO;
       K = STAT(ACTIVE,12);
       IHOLD = FLOOR(K/1000);
       HEHOLD = FLOOR((K-(IHOLD*1000))/100);
       BP = K - (1000 * IHOLD) - (100 * HEHOLD);
    IF IHOLD > 2 & HEHOLD < 1 & BP > 4
       & (25-STAT(PASSIVE,7) < STAT(ACTIVE,10)
       | STAT(PASSIVE,9) > 0)
    THEN RETURN(4);  END;
    IF RPA <= VBACKRPA THEN DO;
       IF STAT(PASSIVE,5) < 3 THEN RETURN(5);
    END;
    RETURN(OLD_STRAT);
END GETSTRT;
ADMRE:
    ENTRY; GO TO TOP;
ADMDI:
    ENTRY;
    PUT SKIP(2) LIST('ENTER 1 FOR PRESENT COEF, 2 FOR BOTH'||
        ' OLD AND NEW');
    GET LIST(I3);
    PUT SKIP(2) LIST('ENTER CODES FOR WHILE,BLACK(0=NONE)');
    GET LIST(I1,I2);
    PUT SKIP(2) LIST('DATA TO FILE?, 1 OR 0');
    GET LIST(I4);
```

```
   CALL DISPLAY(I1,I2,I3,I4);
   RETURN;
ADMSO:
    ENTRY;
   CALL SORTCT;
   PUT SKIP(2) LIST('SORT IS COMPLETED');
   RETURN;
ADMLO:
   ENTRY;  CALL FIND; RETURN;
ADMLI:
   ENTRY; CALL PUTLIST; RETURN;
ADMSW:
   ENTRY;
   CALL SWAP;  PUT SKIP(2) LIST('SWAP COMPLETED'); RETURN;
ADMST:
   ENTRY;
   PUT SKIP(2) LIST('ENTER BOARD CODE,STRATEGY CODES(W,B), AND '||
      'COEF CODES(W,B)(0 = N.A.)');
   GET LIST(I1,I2,I3,I4,I5);
   CALL STORE(I1,I2,I3,I4,I5);   RETURN;
    /*****************************************************************/
    /*                    SUBROUTINE STORE                           */
    /*****************************************************************/
STORE:   PROC(BDCODE,SCODEW,SCODEB,CCODEW,CCODEB);
DCL (BDCODE,SCODEW,SCODEB,CCODEW,CCODEB,PUTCODE,P) FIXED BIN(15);
DCL (FBOARD,FCOEF,FSTRAT) FILE STREAM ENV(F(80));
   IF BDCODE = 0 THEN GO TO ST20;
   OPEN FILE(FBOARD) OUTPUT;
   PUT FILE(FBOARD) LIST(BDCODE, (GB(I) DO I = 24 TO 1 BY -1));
   CLOSE FILE(FBOARD);
ST20:  IF SCODEW = 0 THEN GO TO ST30;
   PUTCODE = SCODEW;  P = 1 ;
ST21:  OPEN FILE(FSTRAT) OUTPUT;
   PUT SKIP FILE(FSTRAT) LIST(PUTCODE);
   DO I = 1 TO 7;  DO J = 1 TO 3;
      PUT SKIP FILE(FSTRAT) LIST
            ((STRAT(P,I,J,K) DO K = 1 TO 10));  END; END;
   CLOSE FILE(FSTRAT);
   IF P = 2 THEN GO TO ST40;
ST30:    IF SCODEB = 0 THEN GO TO ST40;
   PUTCODE = SCODEB;  P = 2;
   GO TO ST21;
ST40:  IF CCODEW = 0 THEN GO TO ST50;
   PUTCODE = CCODEW;   P = 1;
ST41:   OPEN FILE(FCOEF) OUTPUT;
   PUT FILE(FCOEF) EDIT(PUTCODE)(COL(1),F(2));
   DO I = 1 TO 7;  DO J = 1 TO 3;
      PUT SKIP FILE(FCOEF) LIST((C(P,I,J,K) DO K = 1 TO 5));
```

```
      PUT SKIP FILE(FCOEF) LIST((C(P,I,J,K) DO K = 6 TO 10));
   END; END;
   CLOSE FILE(FCOEF);
   IF P = 2 THEN GO TO ST60;
ST50:   IF CCODEB = 0 THEN GO TO ST60;
   PUTCODE = CCODEB;  P = 2;
   GO TO ST41;
ST60:  RETURN;
END STORE;
    /*****************************************************************/
    /*                    SUBROUTINE SWAP                            */
    /*****************************************************************/
SWAP:  PROC;
DCL TC FIXED BIN(15), TT FIXED BIN(31,15);
   DO I2 = 1 TO 7; DO I3 = 1 TO 3; DO I4 = 1 TO 10;
          TT = STRAT(1,I2,I3,I4);
          IF TT = 0 THEN GO TO I3_END;
          TC = C(1,I2,I3,I4);
          C(1,I2,I3,I4) = C(2,I2,I3,I4);
          STRAT(1,I2,I3,I4) = STRAT(2,I2,I3,I4);
          C(2,I2,I3,I4) = TC;
          STRAT(2,I2,I3,I4) = TT;
  END;
I3_END: END;
END;
END SWAP;
    /*****************************************************************/
    /*                    SUBROUTINE DISPLAY                         */
    /*****************************************************************/
DISPLAY:  PROC(WHTCODE,BLKCODE,TYPE,IFLAG);
DCL (WHTCODE,BLKCODE,TYPE,CODE,TESTCODE,P,IFLAG) FIXED BIN(15),
   (FCOEF,FSTRAT) FILE STREAM ENV(F(80)),
   (HEADING1,HEADING2) CHAR(80),
   PS CHAR(8), PT CHAR(7), (BO,BN,WO,WN) FIXED BIN(31,15),
   T  FIXED BIN(15),
   BUFF CHAR(80) VAR,
   GAMOUT FILE STREAM ENV(F(80)),
   STRATSTRING CHAR(56) INIT CALL SSTR,
   TACTICSTR CHAR(350) INIT CALL TSTR,
   (HOLDC(2,7,3,10) FIXED BIN(31,15),HOLDS(2,7,3,10) FIXED BIN(15))
   CONTROLLED;
SSTR:  PROC;
STRATSTRING = 'POSITIONRUN-OFF RUN-DEF BLOCKINGBACK    BO-OFF  BO-DEF';
   END;
TSTR:  PROC;
TACTICSTR = 'REARMENCLOSEYRCLOSEHSUNUSED MAXPTS BLDBLCKCATCHERBLOTYTB'||
'FRZRNRSGETHOME8BEFOR7ADVBLOKMAXSAFELONGRUNENTLOW CSTOOUTBOMOST MXBOODSHI2EVEN'
||'TOLOWPTCOVER  DISTRIBSAVAREAUNUSED UNUSED UNUSED UNUSED UNUSED UNUSED '||
```

```
'HIS 5  HIS 4  HIS 7  YOUR 3 YOUR 4 YOUR 5 YOUR 6 YOUR 7 YOUR 8 YOUR 9 '||
'HITALL HITOUTRHITHSINHITYRINUNUSED ENTERLOENTRHITENTNOHTENTOCVR'||
'ENTNOCV RISK  ';  RETURN;
END;
HEADING1 = '                                        WHITE'||
           '                                BLACK';
HEADING2 = '      STRATEGY   TACTIC          NEW           OLD'||
    '                NEW            OLD';
   IF IFLAG = 1 THEN OPEN FILE(GAMOUT) OUTPUT;
   IF TYPE ¬= 2 THEN GO TO DS8;
   ALLOCATE HOLDS;
   OPEN FILE (FSTRAT) INPUT;
   P = 1;
   IF WHTCODE = 0 THEN GO TO DS1;
   TESTCODE = WHTCODE;
DS2:  GET FILE(FSTRAT) LIST(CODE);
   IF CODE = TESTCODE THEN DO;
   GET FILE(FSTRAT) LIST
   ((((HOLDS(P,I,J,K) DO K = 1 TO 10) DO J=1 TO 3) DO I=1 TO 7));
   GO TO DS1;  END;
   ELSE DO;
      GET FILE(FSTRAT) EDIT(BUFF) (SKIP(21),COL(1),A(80));
      GO TO DS2;  END;
DS1:  IF BLKCODE = 0 THEN GO TO DS3;
   ELSE IF P = 1 THEN DO;
      P = 2;  TESTCODE = BLKCODE;
      GO TO DS2;  END;
  /*  GET COEFFICIENTS   */
DS3:  CLOSE FILE(FSTRAT);
   OPEN FILE(FCOEF) INPUT;
   ALLOCATE HOLDC;
   P = 1;
   IF WHTCODE = 0 THEN GO TO DS5;
   TESTCODE = WHTCODE;
DS4:  GET FILE(FCOEF) LIST(CODE);
   IF CODE = TESTCODE THEN DO;
      GET FILE(FCOEF) LIST
       ((((HOLDC(P,I,J,K) DO K=1 TO 10) DO J=1 TO 3) DO I=1 TO 7));
      GO TO DS5; END;
   ELSE DO;
      GET FILE(FCOEF) EDIT(BUFF) (SKIP(42),COL(1),A(80));
      GO TO DS4;  END;
DS5:  CLOSE FILE(FCOEF);
   IF BLKCODE = 0 THEN GO TO DS8;
   ELSE IF P = 1 THEN DO;
      P = 2;  TESTCODE = BLKCODE;
      GO TO DS4; END;
DS8:
```

```
BUFF = HEADING1;  PUT SKIP EDIT(BUFF)(PAGE,LINE(3),A(80));
BUFF = HEADING2; PUT SKIP(2) LIST(BUFF);
DO I = 1 TO 7;
BO,BN,WO,WN = 0;
   DO J = 1 TO 3;
      DO K = 1 TO 10;
         IF WHTCODE = 0 THEN DO;
           WO,WN = 0; GO TO DS9;  END;
         ELSE T = STRAT(1,I,J,K);
           GO TO DS11;
      DS9:  IF BLKCODE = 0 THEN DO;
           BO,BN = 0; GO TO DS11;  END;
      DS10:  T = STRAT(2,I,J,K);
      DS11:   IF T = 0 THEN GO TO END_J;
              ELSE PT = SUBSTR(TACTICSTR,((T*7)-6),7);
      DS12:  IF K = 1 THEN IF J = 1 THEN DO;
              PS = SUBSTR(STRATSTRING,((8*I)-7),8);
              GO TO DS13;  END;
             PS = '        ';
      DS13: IF WHTCODE = 0 THEN GO TO DS15;
      WN = C(1,I,J,K);
      IF TYPE = 1 THEN DO;
         WO = 0;  GO TO DS15;  END;
      DO JJ = 1 TO 3;  DO KK = 1 TO 10;
         LL = HOLDS(1,I,JJ,KK);
         IF LL = T THEN DO;
            BO = HOLDC(1,I,JJ,KK);
            GO TO DS15;  END;
       END; END;
      DS15:  IF BLKCODE = 0 THEN GO TO DS16;
         DO JJ = 1 TO 3;  DO KK = 1 TO 10;
             LL = STRAT(2,I,JJ,KK);
             IF LL = T THEN DO;
         BN = C(2,I,JJ,KK);
               GO TO DS16;  END;
          END; END;
      IF TYPE = 1 THEN DO;  BO = 0;  GO TO DS16; END;
         DO JJ = 1 TO 3;  DO KK = 1 TO 10;
            LL = HOLDS(2,I,JJ,KK);
            IF LL = T THEN DO;
              BO = HOLDC(2,I,JJ,KK);
              GO TO DS16;  END;
         END; END;
DS16:  BUFF = PS||'   '||PT||'   '||WN||WO||BN||BO;
     IF IFLAG = 1 THEN PUT FILE(GAMOUT) EDIT(BUFF)(SKIP(1),COL(1),A(80));
      ELSE PUT EDIT (BUFF)(SKIP(1),COL(1),A(80));
        END_K: END;
     END_J: ;END;
```

```
   END_I: ;END;
   IF TYPE = 2 THEN DO; FREE HOLDS; FREE HOLDC;  END;
   IF IFLAG = 1 THEN CLOSE FILE(GAMOUT);
END DISPLAY;
    /*****************************************************************/
    /*                    SUBROUTINE SORTCT                          */
    /*****************************************************************/
SORTCT:  PROC;
DCL(MVR,STRT,LVL,T,LIST_TOP,NEXT_MAX,PTR,SWAPT,TEMPT(30))
   FIXED BIN(15),
   (TEMPC(30), MAXC,SWAPC) FIXED BIN(31,15);
   TEMPT = 0;  TEMPC = 0;
   DO MVR = 1 TO 2;
      DO STRT = 1 TO 7;
      /* RETREIVE COEFFICIENTS AND TACTICS FOR MVR AND STRATEGY */
         DO LVL = 1 TO 3;
            DO T = 1 TO 10;
               I = (10* LVL) + T - 10;
               TEMPT(I) = STRAT(MVR,STRT,LVL,T);
               IF TEMPT(I) = 0 THEN GO TO LVL1_END;
               TEMPC(I) = C(MVR,STRT,LVL,T);
             T1_END: END;
         LVL1_END: END;
      /* SORT THE COEFFICIENTS AND TACTICS IS ASCENDING ORDER  */
         DO LIST_TOP = 1 TO 29 WHILE(TEMPT(LIST_TOP) ¬= 0);
            MAXC = TEMPC(LIST_TOP);
            NEXT_MAX = LIST_TOP;
      SCAN: DO PTR = LIST_TOP - 1 TO 30 WHILE(TEMPT(PTR) ¬= 0);
               IF TEMPC(PTR) ¬= 0 THEN
                 IF TEMPC(PTR) > TEMPC(NEXT_MAX) THEN
                   NEXT_MAX = PTR;
            END;
            IF NEXT_MAX = LIST_TOP THEN GO TO LIST_END;
            SWAPC = TEMPC(LIST_TOP);
            SWAPT = TEMPT(LIST_TOP);
            TEMPC(LIST_TOP) = TEMPC(NEXT_MAX);
            TEMPT(LIST_TOP) = TEMPT(NEXT_MAX);
            TEMPC(NEXT_MAX) = SWAPC;
            TEMPT(NEXT_MAX) = SWAPT;
         LIST_END: END;
      /* REPLACE THEN COEFFICENTS AND TACTICS  */
      DO LVL = 1 TO 3;
         DO T = 1 TO 10;
            I = (10 * LVL) + T - 10;
            IF TEMPT(I) = 0 THEN GO TO LVL2_END;
            STRAT(MVR,STRT,LVL,T) = TEMPT(I);
            C(MVR,STRT,LVL,T) = TEMPC(I);
          T2_END: END;
```

```
      LVL2_END: END;
      STRT_END: END;
   MVR_END: END;
END SORTCT;
    /*****************************************************************/
    /*                    SUBROUTINE FIND                            */
    /*****************************************************************/
FIND: PROC;
DCL M(4) FIXED BIN(15);
   PUT SKIP LIST('ENTER MOVE TO BE LOCATED');
   GET LIST ((M(I) DO I = 1 TO 4));
   IF PASSIVE = 2 THEN M(*) = 25 - M(*);
   DO I = 1 TO KEEPCOUNT;
      DO J = 1 TO 4;
         IF M(J) ¬= LIST(I,J) THEN GO TO END_ILUP;
      END;
      PUT SKIP(2) LIST('MOVE POSITION = '||I||'  MOVE SCORE = '||X(I));
      RETURN;
   END_ILUP:  END;
   PUT SKIP(2) LIST(' MOVE CAN NOT BE LOCATED ');
   RETURN;
END FIND;
    /*****************************************************************/
    /*                    SUBROUTINE PUTLIST                         */
    /*****************************************************************/
PUTLIST:  PROC;
DCL M(4) FIXED BIN(5);
   PUT SKIP(2) LIST ('ENTER NO. OF MOVES YOU WANT LISTED(99=ALL)');
   GET LIST(J);
   PUT SKIP(2) LIST('MOVE NO.                    MOVES'||
'                          SCORE');
   DO I = 1 TO J WHILE(I <= KEEPCOUNT);
      DO K = 1 TO 4;  M(K) = LIST(I,K); END;
      IF PASSIVE = 2 THEN M(*) = 25 - M(*);
      PUT SKIP LIST(I||M(1)||M(2)||M(3)||M(4)||X(I));
    END;
RETURN;
END PUTLIST;
END ADMIN1;


     /*****************************************************************/
     /*                                                               */
     /*                    M O D U L E   S E T U P                    */
     /*                                                               */
     /*****************************************************************/

 SETUP: PROC(BATCH,BCODE,SCODEW,SCODEB,CCODEW,CCODEB,M,MV,WB,GB,S,
```

```
            STR,C,CS,PE,PH,OPN,BLOT,F,T,TRACE,CONT);
/* SETUP READS IN THE SPECIFIED GAME BOARD, STRATEGY AND
   COEFFICIENT SETS AND INITIALIZES THE GAME STATISTICS  */
DCL  (BCODE,  /*  BOARD SET UP, 1 = STANDARD  */
     SCODEW,SCODEB, /*  SET OF STRATEGIES FOR WHITE AND BLACK  */
     CCODEW,CCODEB,  /*  SET OF COEFFICIENTS(0: ALL C'S = 1) */
     M,  /*  THE ACTIVE PLAYER, 1=WHT, 2=BLK  */
     MV(4),  /*  THE CURRENT MOVE LIST   */
     GB(24),  /*  THE GAME BOARD  */
     WB(24),  /*  THE WORKING GAME BOARD   */
     S(2,0:12),  /*  GAME STATISTICS   */
     STR(2,10,3,10),  /*  STRATEGY MATRIX  */
     CS(2),  /*  CURRENT STRATEGY OF PLAYERS   */
     CD,  /*  WORKING  VARIABLE   */
      OPN(27,5))  /* TABLE OF OPENING MOVES  */ FIXED BIN(15),
      C(2,10,3,10)  /* MATRIX OF COEFFICIENTS  */ FIXED BIN(31,15),
      (PE(0:6),  /* PROBABILITY TABLE FOR ENTRY  */
      PH(6,0:5))  /*  PROB TABLE FOR HITS */
                    FIXED BIN(15,10),
      BLOT(2,24) BIT(1),  /*  LOCATION OF BLOTS    */
        BATCH BIT(1),CONT BIT(1),
      TRACE BIT(1),    /*   1=PROVIDE TRACE     */
      ((T,F)(2)) BIT(24),  /* TO/FROM WHICH MOVES ARE LEGAL  */
      BUFF CHAR(78) VAR,
        FLAG BIT(1), CODE FIXED BIN(15),
      (FBOARD , FSTRAT, FCOEF, FTABLE) FILE STREAM
           ENVIRONMENT (F(80)) ;
DCL MVIMP ENTRY(,FIXED BIN(15),FIXED BIN(15),,,,,FIXED BIN(31,15),
     ,BIT(1),,,);
DCL(GETC,GETS) ENTRY(FIXED BIN(15),FIXED BIN(15)),
     SETC ENTRY(FIXED BIN(15));
  /*  READ IN GAME BOARD   */
S0:    OPEN FILE(FBOARD) INPUT;
       GET FILE(FBOARD) EDIT(CD,BUFF)(COL(1),F(2),A(78));
       IF CD ¬= BCODE THEN GO TO S0;
       GET STRING(BUFF) LIST((GB(I) DO I = 24 TO 1 BY -1));
       CLOSE FILE(FBOARD);
       IF CONT THEN GO TO S10;
S1:    CALL GETS(SCODEW,1);
       IF SCODEB = SCODEW THEN STR(2,*,*,*) = STR(1,*,*,*);
       ELSE CALL GETS(SCODEB,2);
S2:    IF CCODEW = 0 THEN CALL SETC(1);
       ELSE CALL GETC(CCODEW,1);
       IF CCODEB = CCODEW THEN C(2,*,*,*) = C(1,*,*,*);
       ELSE IF CCODEB = 0 THEN CALL SETC(2);
       ELSE CALL GETC(CCODEB,2);
 S5:   /*  READ IN PE, PH, OPEN   */
       GET FILE(FTABLE) LIST( (PE(I) DO I = 0 TO 6));
```

```
S6:    GET FILE(FTABLE) LIST(((PH(I,J) DO J = 0 TO I-1) DO I = 6 TO 1 BY
         -1));
S7:    GET FILE(FTABLE) LIST ((OPN(I,1),(OPN(I,J) DO J = 2 TO 5)
          DO I = 1 TO 27));
S8:    /*     INITIALIZE STRATEGIES  */
       IF CONT THEN DO; CS = 1;  GO TO S10;      END;
       PUT SKIP LIST('     ENTER INITIAL WHITE AND BLACK STRATEGIES:');
        PUT SKIP LIST('      1=POSITION, 2=RUN-OFF, 3=RUN-DEF, 4=BLOCKING, 5=BACK
        ', 6=BEAR-OFF, 7=BEAR-DEF');
       GET LIST( CS(1),CS(2));
S9:    /* NON-STANDARD SET UP?  */
       IF BCODE ¬= 1 THEN DO;
       CALL MVIMP(M,0,0,S,WB,GB,MV,0.0,BLOT,'0'B,T,F,TRACE);
       RETURN; END;
S10:   /* INITIALIZE STATISTICS FOR STANDARD GAME  */
       S = 0;
       BLOT = '0'B;
       T(1),T(2) = '111110101111011111111110'B;
       F(1),F(2) = '100000000001000010100000'B;
    /*   BEAR-OFF DISTANCE   */
       S(1,2) = 77;
    /* MEN LEFT TO BEAR OFF  */ S(1,3) = 15;
    /*  OVERLAP  */  S(1,4) = 96;
    /*  HOME POINTS HELD  /*  S(1,5) = 1;
    /* OPPONENT'S HOME POINTS HELD  */ S(1,6) = 1;
    /* REAR MEN  */  S(1,7) = 24;
    /* NEXT-TO-REAR MEN  */  S(1,8) = 13;
    /*  HEAD OF BLOCK  */  S(1,10) = 5;
    /*  TAIL OF BLOCK  */  S(1,11) = 11;
    /*  BLOCK POTENTIAL  */ S(1,12) = 2008;
       S(2,*) = S(1,*);
RETURN;
     /*****************************************************************/
     /*                      SUBROUTINE  GETS                         */
     /*****************************************************************/
GETS:    PROC(CD,P);
DCL (CD,P,CODE) FIXED BIN(15);
       ON ENDFILE(FSTRAT) BEGIN;
       PUT SKIP LIST('ENDFILE FSTRAT');  GO TO G2;  END;
       OPEN FILE(FSTRAT) INPUT;
       GET FILE(FSTRAT) LIST(CODE);
G1:  IF CODE = CD THEN GET FILE(FSTRAT) LIST
((((STR(P,I,J,K) DO K=1 TO 10) DO J=1 TO 3) DO I=1 TO 7));
       ELSE DO;
         GET FILE(FSTRAT) SKIP(22) LIST(CODE);
         GO TO G1;  END;
G2:  CLOSE FILE(FSTRAT);
   RETURN;
```

```
END GETS;
     /*****************************************************************/
     /*                    SUBROUTINE  GETC                           */
     /*****************************************************************/
GETC:   PROC(CD,P);
   DCL(CD,P,CODE) FIXED BIN(15);
   ON ENDFILE(FCOEF) BEGIN;
      PUT SKIP LIST('ENDFILE FCOEF'); GO TO G4;  END;
   OPEN FILE(FCOEF) INPUT;
   GET FILE(FCOEF) LIST(CODE);
   G3: IF CODE = CD THEN GET FILE(FCOEF) LIST
((((C(P,I,J,K) DO K=1 TO 10) DO J=1 TO 3) DO I=1 TO 7));
   ELSE DO;
      GET FILE(FCOEF) SKIP(43) LIST(CODE);
      GO TO G3;  END;
G4:  CLOSE FILE(FCOEF);
RETURN;
END GETC;
     /*****************************************************************/
     /*                    SUBROUTINE  SETC                           */
     /*****************************************************************/
SETC:   PROC(P);
DCL P FIXED BIN(15);
   DO I = 1 TO 3;
      I1 = 2**(3-I);
      DO J = 1 TO 7;
         DO K = 1 TO 10;
            C(P,J,I,K) = I1;
   END; END; END;
RETURN;
END SETC;
END SETUP;

     /*****************************************************************/
     /*                                                               */
     /*                    M O D U L E   M A S K                      */
     /*                                                               */
     /*****************************************************************/

MASK:           PROC(D1,D2,F,T,FROM,TO,TRACE);
 DCL (T,F) BIT(24) ,
     FROM(4,24) BIT(1) ,
     TO(4,24) BIT(1) ;
    DCL(D1,D2) FIXED BIN(15),
    (TOS,FROMS) (4) BIT(24),
      ( FILL1,FILL2) BIT(6) VAR,
  FILL3 BIT(12) VAR,
    ZEROS BIT(6) INITIAL('000000'B);
```

```
   DCL (TRACE,D) BIT (1), BUFF CHAR(24);
   /*  FROM AND TO ARE MATRICES WHICH INDICATE FROM WHICH AND TO WHICH
  LEGAL MOVES MAY BE MAKE.  THEY ARE CONSTRUCTED USING : T, A STRING
   INDICATING ALL POINTS ON THE BOARD WHICH CAN LEGALLY RECEIVE A PIECE;
   F,  A STRING INDICATING ALL POINTS ON THE BOARD WHICH CAN SUPPLY A
   PIECE;  D1 AND D2 ARE THE ROLLS OF THE DICE.  NOTE THAT T AND F
   ARE READ(IN THE SENSE OF MOVEMENT OF PIECES) FROM LEFT TO RIGHT
   WHERE THE LEFT END IS ACTUALLY PT 24 AND THE RIGHT END PT1.   *.*/
   FILL1 = SUBSTR(ZEROS,1,D1);
 IF D1 = D2 THEN D = '1'B; ELSE D = '0'B;
 K = 24 - D2;
 J = 24 - D1;
 TOS(1) = (FILL1||F) & T;
 FROMS(1) = SUBSTR(TOS(1),D1 + 1,J)||FILL1;
 FILL2 = SUBSTR(ZEROS,1,D2);
 L = D1 + D2 + 1;
 FILL3 = FILL1||FILL2;
 TOS(3) = (FILL2||TOS(1)) & T;
 FROMS(3) = SUBSTR(TOS(3),L,25-L)||FILL3;
 IF D THEN GO TO MSK1;
 TOS(2) = (FILL2||F) & T;
 FROMS(2) = SUBSTR(TOS(2),D2+1,K)||FILL2;
 TOS(4) = (FILL1||TOS(2)) & T;
 FROMS(4) = SUBSTR(TOS(4),L,25-L)||FILL3;
 MSK1: L1 = 1; L2 = 3;
 TO = '1'B; FROM = '0'B;
 MOVE:  DO I = L1, L2;
                DO J = 1 TO 24;
                     IF SUBSTR(TOS(I),J,1) = '0'B THEN TO(I,25-J) = '0'B;
                     IF SUBSTR (FROMS(I),J,1) = '1'B THEN
                          FROM(I,25-J) = '1'B;
                     END;
     END;
 IF L1 = 2 THEN GO TO OUT;
 IF D THEN DO;
          TO(2,*) = TO(1,*);
          TO(4,*) = TO(3,*);
          FROM(2,*) = FROM(1,*);
          FROM(4,*) = FROM(3,*);
          GO TO OUT;  END;
 ELSE DO ; L1 = 2; L2 = 4; GO TO MOVE; END;
OUT: DO;
   IF D1 = 0 THEN DO;
      FROM(1,*), TO(1,*) = '0'B;  GO TO OUT1; END;
   ELSE IF D2 = 0 THEN DO;
      FROM(2,*), TO(2,*) = '0'B; END;
   ELSE GO TO OUT2;
   OUT1: TO(3,*), TO(4,*), FROM(3,*), FROM(4,*) = '0'B;
```

```
  END;
 OUT2:  IF TRACE THEN DO;
      DO I = 1 TO 2;
           IF I = 1 THEN PUT SKIP LIST('.....FROM MATRIX:');
           ELSE PUT SKIP LIST('.....TO MATRIX:');
           DO J = 1 TO 4;
                IF I = 1 THEN BUFF = STRING(FROM(J,*));
                ELSE BUFF = STRING(TO(J,*));
                PUT SKIP LIST(BUFF);  END;
      END;
   END;
 RETURN;
END MASK;

     /*****************************************************************/
     /*                                                               */
     /*                        M O D U L E   R O L L                  */
     /*                                                               */
     /*****************************************************************/

ROLL:          PROC(S,D1,D2);
 /* ROLL RETURNS TWO RANDOM INTEGERS ON THE INTERVAL (1,6)
    SUCH THAT  D1 > D2  */
   DCL S FIXED BIN(31),
       (D1,D2,DT) FIXED BIN(15),
       RANDOM ENTRY(FIXED BIN(31)) RETURNS(FLOAT BIN(21));
   D1 = FLOOR(6 * RANDOM(S)) + 1;
   D2 = FLOOR(6 * RANDOM(S)) + 1;
   IF D2 > D1 THEN DO; DT = D1;D1 = D2;D2 = DT; END;
   RETURN;
 /* RANDOM RETURNS A PSEUDO-RANDOM NUMBER ON THE INTERVAL
   0 TO 1.  IF S = 0 THEN THE NEXT VALUE IN THE SEQUENCE IS
   TAKEN.  IF S > 0 THEN THE SEQUENCE STARTS AT A VALUE DETERMINED
   BY S. IF S IS NEGATIVE THEN THE SEQUENCE STARTS AGAIN AS IN
   THE CALL RANDOM(0). */
   RANDOM:     PROC(S) FLOAT BIN(21);
   DCL S FIXED BIN(31), RV FIXED BIN(31) STATIC INITIAL(45),
         R FLOAT BIN(21) STATIC;
   IF S < 0 THEN RV = 3587;
      ELSE IF S > 0 THEN RV = S;
      RV =  MOD(RV*3587,524288);
      R=RV;
      RETURN(MOD(R,32768)/32768);
   END RANDOM;
END ROLL;
```

```
/*****************************************************************/
/*                                                               */
/*                      M O D U L E   M V G E N                  */
/*                                                               */
/*****************************************************************/

MVGEN:   PROC(M,S,L,A,D1,D2,BD,TMOVE,STRAT,STAT,BLOT,FROM,TO,ENTER,
                TRACE);
  /*  MVGEN GENERATES LEGAL MOVES GIVEN:
      M, THE PLAYER WHOSE MOVE IT IS(1=WHITE,2=BLACK)
      S, THE MOVER'S CURRENT STRATEGY
      L, THE PRIORITY LEVEL OF TACTICS
      A, THE NO. OF AVAILABLE DICE UNITS(1 OR 2)
      D1,D2,  THE CURRENT DICE ROLL(D1 >= D2)
      BD, THE CURRENT BOARD CONFIGURATION  */
DCL (M,S,L,A,D1,D2,BD(24),IR,II,J,K,LL,MM) FIXED BIN(15),
     (STRAT(2,10,3,10),STAT(2,0:12)) FIXED BIN(15) ,
      BLOT(2,24) BIT(1) ,
      G(0:50) LABEL;
DCL  (MVTO,MVFM,MVHIT) ENTRY (FIXED BIN(15), FIXED BIN (15)),
      V1 FIXED BIN (15) INITIAL (1) STATIC,
      V2 FIXED BIN (15) INITIAL (2) STATIC;
DCL (((FROM,TO) (4,24)) ,TRACE,ENTER) BIT(1);
DCL (DOUBLES,ON,ONEREAR) BIT (1);
DCL     1 TMOVE,
            2 REG, 3 (DH,B12,B21,CNV,FRK,SH1,SH2) BIT(24),
            2 BOBM, 3 (B1,B2,B3,B4,B5,B6,B7,B8) BIT(6),
            2 ED, 3 (ED1,ED2) BIT(2);
   IF L = 1 THEN DO;
      DH,B12,B21,CNV,FRK,SH1,SH2 = (24) '0'B;
      B1,B2,B3,B4,B5,B6,B7,B8 = (6) '0'B;
      ED1,ED2 = '00'B; END;
   IF S = 99 THEN DO;
      DO I = D2 + 1 TO 24;
         CALL MVFM(A,I); END;  RETURN; END;
   ON = '1'B;
   /* SET UP GUIDE: EXTRACT APPROPRIATE SET OF TACTICS FROM  STRAT */
   IF D1 = D2 THEN DOUBLES = '1'B;  ELSE DOUBLES = '0'B;
 /*THE FOLLOWING LOOP CONTROLS FLOW TO INDIVIDUAL MOVE GENERATORS */
   II = 1;
   IF M= 1 THEN MM = 2;  ELSE MM = 1;
   GTOP: LL = STRAT(M,S,L,II);
   IF LL < 0 | LL > 50 THEN DO;
      PUT SKIP LIST('BAD LABEL IN MVGEN'||LL);GO TO GENLOOP; END;
      IF ENTER THEN IF LL-40 < 5 THEN GO TO GENLOOP;
   XXX1: IF TRACE THEN DISPLAY(' GEN. MOVES FOR TACTIC'||LL);
   GO TO G(LL);
   GENLOOP: II = II + 1; IF II = 11 THEN GO TO G(0);
```

```
  GO TO GTOP;
G(0): RETURN;
G(4):G(13):G(8):G(9):G(14):G(15):G(50):G(18):G(19):G(20):
     GO TO GENLOOP;
  G(1): /* BRING UP OUTERMOST MEN */
     J = STAT(M,7);
     CALL MVFM(A,J);
  J = STAT(M,8);  IF J > 0 THEN CALL MVFM(A,J);
     GO TO GENLOOP;
  G(2): /* CLOSE YOUR BOARD(INNER TABLE) */
     DO I = 1 TO 6;
          CALL MVTO(A,I); END;
     GO TO GENLOOP;
  G(3): /* CLOSE HIS BOARD(INNER TABLE)  */
     DO I = 19 TO 24;
        CALL MVTO(A,I); END;
  GO TO GENLOOP;
  G(5): /* OCCUPY MAX NO OF POINTS. FIND PTS WITH MORE 2 MEN
           AND MOVE THOSE */
        DO I = 2 TO 24;
           IF BD(I) = 3 THEN DO;
              CALL MVFM(V1,I);
              GO TO G51; END;
           IF BD(I) > 3 THEN DO;
              IF A = 2 THEN CALL MVFM(V2,I);
                 ELSE CALL MVFM(V1,I); END;
        G51: END; GO TO GENLOOP;
  G(6): /* BUILD BLOCK */
     J=STAT(M,10); /* HEAD OF BLOCK AREA */
     K=STAT(M,11); /* TAIL OF BLOCK AREA */
  DO I = J TO K;
     IF BD(I) < K THEN CALL MVTO(A,I); END;
     GO TO GENLOOP;
   /* DEPLOY 'CATCHERS', BLOTS WHICH ONE WANTS HIT IN ORDER
           TO BUILD A BACK GAME.  DEPLOY IN REGION FORWORD OF
           BACK BLOCK IN OUTER TABLES */
  G(7): J=STAT(M,10);    /*  HEAD OF BLOCK    */
     DO I = 4 TO J-1;
        IF BD(I)=0|BD(I)=-1 THEN CALL MVTO(V1,I);/*OPEN SPOT*/
        IF BD(I)=2 THEN CALL MVFM(V1,I);/* UNCOVER HELD PTS */
        IF BD(I)=3  THEN IF  A=2 THEN CALL MVFM(V2,I);
     END;
  GO TO GENLOOP;
  G(10): /* MOVE MEN TO INNER TABLE */
     DO I = 1 TO 6;
        IF BD(I)>-2 THEN CALL MVTO(A,I); END;
  GO TO GENLOOP;
  G(11): /* BRING IN 8PT-MEN IN PREFERENCE TO 7PT-MEN */
```

```
   DO I = 7,8; CALL MVFM(A,I); END;
   GO TO GENLOOP;
G(12): /* ADVANCE BLOCK */
   J = STAT(M,10); /* HEAD OF BLOCK */
   DO I = J-1 TO J-3; CALL MVTO(A,I);END;
   GO TO GENLOOP;
G(16): /* MOVE MEN UP FROM COMF STAT(PT 12) */
   CALL MVFM(A,12); GO TO GENLOOP;
G(21):G(44):  /* COVER ALL BLOTS(21),OR THREATENED BLOTS(44)   */
   IF LL = 21 THEN J = 24; ELSE J = 25-STAT(MM,7);
   DO I = 1 TO J;
      IF BLOT(M,I) THEN DO;
         CALL MVTO(A,I); /* COVER FROM REAR */
         CALL MVFM(A,I); /* MOVE UP BLOT */
   END;END;
   GO TO GENLOOP;
G(22): /* DISTRIBUTE MEN IN YOUR  OUTER TABLE */
   DO I = 7 TO 12;
      IF BD(I) < 2 THEN CALL MVTO(A,I);  END;
   GO TO GENLOOP;
G(23): /* BRING MEN INTO SAFE AREA BEHIND OPPONENTS
          REARMOST MEN */
   J = 25-STAT(MM,7);
   IF J>24 THEN GO TO GENLOOP;
   DO I= 1 TO J-1; CALL MVTO(A,I);END;
   GO TO GENLOOP;
G(30):G(31):G(32) :G(33):G(34):G(35):G(36):G(37):G(38):G(39):
   /* OCCUPY KEY POINTS: YOUR 3-9 PTS;HIS 4,5,7 PTS */
   IF LL>32 THEN DO;
      J=LL-30; CALL MVTO(A,J);GO TO GENLOOP;END;
   ELSE IF LL=30 THEN CALL MVTO(A,20);
   ELSE IF LL=31 THEN CALL MVTO(A,21);
   ELSE IF LL=32 THEN CALL MVTO(A,18); GO TO GENLOOP;
G(40):G(41):G(42):G(43):
   IF LL = 40 THEN DO;  J1 = 1;  J2 = 24;  END;
   ELSE IF LL = 41 THEN DO; J1 = 7;  J2 = 18;  END;
   ELSE IF LL = 42 THEN DO; J1 = 1; J2 = 6; END;
   ELSE IF LL = 43 THEN DO;  J1 = 19; J2 = 24;  END;
   DO I = J1 TO J2;
      IF BLOT(MM,25-I) THEN CALL MVHIT(A,I);   END;
   GO TO GENLOOP;
G(45):G(46):G(47):G(48):G(49):  /* ENTER BOARD FROM BAR  */
   IF ENTER = '0'B THEN GO TO GENLOOP;
   J = 25-D1 ; K = 25-D2;
   IF BD(J) >-2 THEN SUBSTR(ED1,1,1) = ON;
   IF ¬DOUBLES THEN IF BD(K) >-2 THEN SUBSTR(ED2,1,1) = ON;
   IF A = 2 THEN DO;
   IF BD(J) >-2 THEN DO;     /* A = 2 */
```

```
      IF DOUBLES THEN DO;
         SUBSTR(ED1,2,1) = ON;
         GO TO GENLOOP; END;
      IF BD(K) >-2 THEN DO; SUBSTR(ED1,1,1) = ON;
         SUBSTR(ED2,1,1) = ON;  END;
         ELSE SUBSTR(ED1,1,1) = ON;
         GO TO GENLOOP;
   END;
   IF BD(K) >-2 THEN DO;
      SUBSTR(ED2,1,1) = ON;
      IF DOUBLES THEN SUBSTR(ED2,2,1) = ON;
END; END;
GO TO GENLOOP;
G(17):     /*  BEAR OFF   */
   DO I = 1 TO 6;  CALL MVFM(A,I);  END;
   REAR,NEXTREAR = 0;
   DO I = 6 TO 1 BY -1 WHILE(REAR = 0);
      IF BD(I) > 0 THEN REAR = I; END;
   IF BD(REAR) = 1 THEN ONEREAR = '1'B;
      ELSE ONEREAR = '0'B;
   IF ¬ONEREAR THEN NEXTREAR = REAR;
      ELSE DO; DO I = REAR-1 TO 1 BY -1 WHILE(NEXTREAR=0);
         IF BD(I) > 0 THEN NEXTREAR = I;  END;  END;
   IF D1 = 0 THEN GO TO G171;
  /*  BEAROFF USING BOTH D1 AND D2   */
G170:  IF BD(D1) > 0 THEN SUBSTR(B1,D1,1) = ON;
   ELSE IF REAR < D1 THEN SUBSTR(B1,REAR,1) = ON;
   IF D2 = 0 THEN GO TO GENLOOP;
   IF DOUBLES THEN DO;
      IF BD(D1) >= 2 THEN SUBSTR(B2,D1,1) = ON;
      ELSE IF NEXTREAR ¬= 0 THEN IF
      NEXTREAR < D1 THEN SUBSTR(B2,NEXTREAR,1) = ON;
      GO TO G172;  END;
G171:  IF BD(D2) > 0 THEN SUBSTR(B2,D2,1) = ON;
   ELSE IF NEXTREAR ¬= 0 THEN IF NEXTREAR < D2
      THEN SUBSTR(B2,NEXTREAR,1) = ON;
  /*  BEAROFF FROM D1 AND MOVE ON D2   */
   IF A = 1 | D1*D2 = 0 THEN GO TO GENLOOP;
G172:  J = D1 + D2;
   IF J < 7 THEN IF FROM(2,J) THEN SUBSTR(B3,J,1) = ON;
   DO I = REAR TO D2+1 BY -1 WHILE(BD(D1) > 0);
      IF FROM(2,I) THEN DO;
         IF I = D1 THEN IF BD(I) < 2 THEN GO TO G1721;
      SUBSTR(B3,I,1) = ON;
      END;
   G1721: ; END;
  /* SPECIAL D2MV, D1-OFF MOVES    */
    IF REAR > D2 + 1  THEN
```

```
      IF NEXTREAR < D1 THEN DO;
   DO I = REAR TO D2 + 1 BY -1;
      IF FROM(2,I) THEN IF I-D2 < D1 THEN DO;
          SUBSTR(B4,I,1) = ON;
          IF I-D2 <= NEXTREAR THEN SUBSTR(B5,NEXTREAR,1) = ON;
   END; END; END;
   IF DOUBLES THEN GO TO GENLOOP;
  /* BEAROFF FROM D2,MOVE UP ON D1   */
   J = D1 + D2;
   IF J < 7 THEN IF FROM(1,J) THEN SUBSTR(B6,J,1) = ON;
   DO I = REAR TO D1+1 BY -1 WHILE(BD(D2) > 0);
      IF FROM(1,I) THEN DO;
      IF I = D2 THEN IF BD(I) < 2 THEN GO TO G173;
      SUBSTR(B6,I,1) = ON;
      END;
  G173: END;
  /* SPECIAL MOVES D1-MV,D2-OFF  */
    IF REAR > D1+1 THEN
      IF NEXTREAR < D2 THEN DO;
   DO I = REAR TO D1+1 BY -1;
      IF FROM(1,I) THEN IF K-D2 < D1 THEN DO;
      SUBSTR(B7,I,1) = ON;
      IF I-D1 <= NEXTREAR THEN SUBSTR(B8,NEXTREAR,1) = ON;
   END; END; END;
GO TO GENLOOP;
     /*****************************************************************/
     /*                    SUBROUTINE  MVFM                           */
     /*****************************************************************/
MVFM:          PROC(M,P);
   DCL(M,P,Q) FIXED BIN(15);
   Q = 25 - P;
   XXXFM: IF TRACE THEN PUT SKIP LIST('IN MVFM;A,P:'||M||P);
   IF BD(P) <= 0 THEN RETURN;
   IF M = 1 | BD(P)= 1 | D2 = 0 | D1 = 0
      THEN DO;
         IF D1 ¬= 0 THEN IF FROM(1,P) THEN SUBSTR(SH1,Q,1)=ON;
         IF DOUBLES THEN RETURN;
         IF D2 ¬= 0 THEN IF FROM(2,P) THEN SUBSTR(SH2,Q,1)=ON;
         RETURN;
      END;
   IF DOUBLES THEN DO;
      IF FROM(1,P) THEN DO;
         SUBSTR(SH1,Q,1)=ON;
         IF BD(P) >= 2 THEN DO;
            SUBSTR(DH,Q,1)=ON;
            IF FROM(3,P) THEN SUBSTR(B12,Q,1)=ON;
         END;
      RETURN;
```

```
      END;
   END;
   IF FROM(1,P) THEN DO;
      SUBSTR(SH1,Q,1)=ON;
      IF FROM(3,P) THEN SUBSTR(B12,Q,1)=ON;
   END;
   IF FROM(2,P) THEN DO;
      SUBSTR(SH2,Q,1)=ON;
      IF FROM(4,P) THEN SUBSTR(B21,Q,1)=ON;
   END;
   IF BD(P) >= 2 & FROM(1,P) & FROM(2,P) THEN DO;
     SUBSTR(FRK,Q,1)=ON;  END;
   RETURN;
END MVFM;
     /*****************************************************************/
     /*                    SUBROUTINE MVTO                            */
     /*****************************************************************/
MVTO:          PROC(M,P);
   DCL(M,P,Q,P1,P2,P3) FIXED BIN(15);
   XXXTO: IF TRACE THEN PUT SKIP LIST('IN MVTO;A,P:'||M||P);
   Q = 25 - P;P1 = Q - D1;P2 = Q - D2;P3 = Q - D1 -D2;
   IF BD(P) < -1 THEN RETURN;
   IF M = 1 | D2 = 0 | D1 = 0 THEN DO;
      IF D1 ¬= 0 THEN DO;
         IF TO(1,P) THEN SUBSTR(SH1,P1,1)=ON;
         IF DOUBLES THEN RETURN;
      END;
      IF D2 ¬= 0 THEN IF TO(2,P) THEN SUBSTR(SH2,P2,1)=ON;
      RETURN;
   END;
   IF TO(3,P) THEN SUBSTR(B12,P3,1)=ON;
   IF TO(4,P) THEN SUBSTR(B21,P3,1)=ON;
   IF DOUBLES & TO(1,P) THEN DO;
      SUBSTR(SH1,P1,1)=ON;
      IF BD(P+D1) >= 2 THEN SUBSTR(DH,P1,1)=ON;
      RETURN;
   END;
   IF TO(1,P) THEN DO;
      SUBSTR(SH1,P1,1)=ON;
      IF TO(2,P) THEN DO;
         SUBSTR(SH2,P2,1)=ON;
         SUBSTR(CNV,Q,1)=ON;
         RETURN;
      END;
   END;
   IF TO(2,P) THEN SUBSTR(SH2,P2,1)=ON;
   RETURN;
END MVTO;
```

```
     /*****************************************************************/
     /*                    SUBROUTINE  MVHIT                          */
     /*****************************************************************/
MVHIT:          PROC(M,P);
   DCL(M,P,Q,P1,P2,P3) FIXED BIN(15);
   XXXHIT: IF TRACE THEN PUT SKIP LIST('IN MVHIT;A,P:'||M||P);
   Q = 25-P;P1=Q-D1; P2 = Q-D2; P3 = Q -D1 -D2;
   IF M = 1 | D2 = 0 | D1 = 0 THEN DO;
      IF D1 ¬= 0 THEN DO;
         IF TO(1,P) THEN SUBSTR(SH1,P1,1)=ON;
         IF DOUBLES THEN RETURN;
      END;
      IF D2 ¬= 0 & TO(2,P) THEN SUBSTR(SH2,P2,1)=ON;
      RETURN;
   END;
   IF DOUBLES THEN DO;
      IF TO(1,P) THEN DO;
         IF BD(P+D1) >= 2 THEN DO;
            SUBSTR(DH,P1,1)=ON;
            RETURN;
         END;
      SUBSTR(SH1,P1,1)=ON;
      RETURN;
      END;
   RETURN;
   END;
   IF TO(3,P) THEN SUBSTR(B12,P3,1)=ON;
   IF TO(4,P) THEN SUBSTR(B21,P3,1)=ON;
   IF TO(1,P) THEN DO;
      SUBSTR(SH1,P1,1)=ON;
      IF FROM(3,P+D1) THEN SUBSTR(B12,P1,1)=ON;
   END;
   IF TO(2,P) THEN DO;
      SUBSTR(SH2,P2,1)=ON;
      IF FROM(4,P+D2) THEN SUBSTR(B21,P2,1)=ON;
      IF TO(1,P) THEN SUBSTR(CNV,Q,1)=ON;
   RETURN;
   END;
END MVHIT;
 END MVGEN;

     /*****************************************************************/
     /*                                                               */
     /*                    M O D U L E   M V L I S T                  */
     /*                                                               */
     /*****************************************************************/

MVLIST:         PROC(A,D,D1,D2,B,L,K,TRACE);
```

```
   DCL (TRACE,D) BIT(1),T BIT(1) INITIAL('1'B),
  1 B,    2 R, 3 (DH,B12,B21,CNV,FRK,SH1,SH2) BIT(24),
          2 BOBM, 3 (B1,B2,B3,B4,B5,B6,B7,B8) BIT(6),
          2 ED, 3 (ED1,ED2) BIT(2),
      NEXT ENTRY(BIT(24),FIXED BIN(15)) RETURNS(FIXED BIN(15)),
      L(50,4) FIXED BIN(15),
      LIST ENTRY(FIXED BIN(15),FIXED BIN(15),FIXED BIN(15),FIXED BIN(15),
                 FIXED BIN(15)),
      (A,K,N,N1,N2,J,J2,M,M1,M2,D1,D2,D3) FIXED BIN(15);
  DCL Z FIXED BIN(15) INIT(0) STATIC,
      Y FIXED BIN(15) INIT(25) STATIC;
 K = K + 1;
 T = '1'B;
 D3 = D1 + D2;
 IF A= 1 | D1*D2 = 0 THEN GO TO L8;
 IF D = '0'B THEN GO TO L3;
   XXX1: IF TRACE THEN PUT SKIP LIST('L2,DH');
      N = INDEX(DH,T);
      DO WHILE(N ¬= 0);
              M = 25-N;
      CALL LIST(K,M,M-D1,M,M-D1);
         N = NEXT(DH,N);END;
  L3: /* LIST D1-D2 BOUNCE MOVES */
   XXX2: IF TRACE THEN  PUT SKIP LIST('L3,B12');
      N = INDEX(B12,T);
      DO WHILE(N¬=0);
         M = 25 - N ;
         CALL LIST(K,M,M-D1,M-D1,M-D3);
         N=NEXT(B12,N);END;
  L4: /* LIST D2-D1 BOUNCE MOVES */
   XXX4: IF TRACE THEN  PUT SKIP LIST('L4,B21');
      IF D THEN GO TO L6;
      N=INDEX(B21,T);
      DO WHILE(N¬=0);
         M = 25 - N ;
         CALL LIST(K,M,M-D2,M-D2,M-D3);
         N=NEXT(B21,N);END;
   L5: /* LIST CONVERGE MOVES */
   XXX5: IF TRACE THEN  PUT SKIP LIST('L5,CNV');
      N=INDEX(CNV,T);
      DO WHILE(N¬=0);
         M = 25 - N ;
         CALL LIST(K,M+D1,M,M+D2,M);
         N=NEXT(CNV,N);END;
XXX51:  IF TRACE THEN PUT SKIP LIST('L51,FRK');
      N = INDEX(FRK,T);
      DO WHILE(N ¬= 0);
         M = 25 - N;
```

```
        CALL LIST(K,M,M-D1,M,M-D2);
        N = NEXT(FRK,N); END;
     GO TO L7;
  L6: /* LIST SH COMBINATIONS FOR DOUBLES */
XXX6: IF TRACE THEN PUT SKIP LIST('L6,SH COMB,D');
      N1=INDEX(SH1,T);
      DO WHILE(N1¬=0);
      M1 = 25 - N1;
         J=M1-D1;
         N2=NEXT(SH1,N1);
         IF N2=N1+D1 THEN IF SUBSTR(B12,N1,1)
            THEN N2=NEXT(SH1,N2);
         DO WHILE(N2¬=0);
            M2 = 25 - N2;
            CALL LIST(K,M1,J,M2,M2-D1);
            N2=NEXT(SH1,N2); END;
         N1=NEXT(SH1,N1);
      END;
      GO TO L11;
  L7: /* LIST ALL D1-D2 COMBINATIONS */
  XXX7: IF TRACE THEN PUT SKIP LIST('L7,SH COMB');
      N1=INDEX(SH1,T);
      DO WHILE(N1¬=0);
         M1 = 25 - N1;
         J=M1-D1;
         N2=INDEX(SH2,T);
      L72: DO WHILE(N2¬=0);
         IF ABS(N1-N2)=D1 THEN DO;
            IF N1<N2 THEN
               IF SUBSTR(B12,N1,1) THEN GO TO L73;
            IF N2>N1 THEN IF SUBSTR(B21,N2,1) THEN GO TO L73;
         END;
      M2 = 25 - N2; J2 = M2 - D2;
       IF M1 = J2 THEN GO TO L73;
       IF M1 = M2 THEN GO TO L73;
      IF J = J2 THEN IF SUBSTR(CNV,25-J,1) THEN GO TO L73;
            CALL LIST(K,M1,J,M2,J2);
            L73: N2=NEXT(SH2,N2); END;
         N1=NEXT(SH1,N1);
      END;
      GO TO L13;
  L8: /* LIST D1 SINGLE HOPS */
  XXX8: IF TRACE THEN  PUT SKIP LIST('L8,D1 SINGLES');
  IF D1 = 0 THEN GO TO L9;
      N=INDEX(SH1,T);
      DO WHILE(N¬=0);
         M = 25 - N ;
         CALL LIST(K,M,M-D1,Z,Z);
```

```
          N=NEXT(SH1,N);END;
        IF SUBSTR(ED1,1,1) THEN CALL LIST(K,Y,25-D1,Z,Z);
        N = INDEX(B1,T);
        IF N¬= 0 THEN CALL LIST(K,N,Z,Z,Z);
    L9: /* LIST D2 SINGLE HOPS */
  XXX9: IF TRACE THEN PUT SKIP LIST('L9,D2 SINGLES');
    IF D2 = 0 THEN GO TO L101;
        N=INDEX(SH2,T);
        DO WHILE(N¬=0);
           M = 25 - N ;
           CALL LIST(K,M,M-D2,Z,Z);
           N=NEXT(SH2,N);END;
        IF SUBSTR(ED2,1,1) THEN CALL LIST(K,Y,25-D2,Z,Z);
        N = INDEX(B2,T);
        IF N¬= 0 THEN CALL LIST(K,N,Z,Z,Z);
      GO TO L101;
   L11: /* LIST ENTER MOVES WITH DOUBLES */
 XXX11: IF TRACE THEN  PUT SKIP LIST('L11,EMD');
        IF SUBSTR(ED1,2,1) THEN
           CALL LIST(K,Y,25-D1,Y,25-D1);
        IF SUBSTR(ED2,2,1) THEN CALL LIST(K,Y,25-D2,Y,25-D2);
    L13: /* BEAR OFF/MOVE COMBINATIONS */
    XXX13: IF TRACE THEN  PUT SKIP LIST('L13,BO-BM COMB');
    IF TRACE THEN PUT SKIP LIST('BO D1 & D2');
    M1 = INDEX(B1,T);
    M2 = INDEX(B2,T);
    IF M1*M2 ¬= 0 THEN CALL LIST(K,M1,Z,M2,Z);
IF TRACE THEN PUT SKIP LIST('BO-D1,MV-D2');
    M2 = INDEX(B3,T);
    DO WHILE(M2 ¬= 0);
        CALL LIST(K,M2,M2-D2,D1,Z);
        M2 = NEXT(B3,M2);  END;
IF TRACE THEN PUT SKIP LIST('MV-D1,BO-D2');
    M1 = INDEX(B6,T);
    DO WHILE(M1 ¬= 0);
        CALL LIST(K,M1,M1-D1,D2,Z);
        M1 = NEXT(B6,M1);  END;
IF TRACE THEN PUT SKIP LIST('SPEC MV-D2,D1-OFF');
    M1 = INDEX(B5,T);
    M2 = INDEX(B4,T);
    DO WHILE(M2 ¬= 0);
        IF M1 = 0 THEN CALL LIST(K,M2,M2-D2,M2-D2,Z);
        ELSE CALL LIST(K,M2,M2-D2,M1,Z);
        M2 = NEXT(B4,M2);  END;
IF TRACE THEN PUT SKIP LIST('SPEC MV-D1,D2-OFF');
    M2 = INDEX(B8,T);
    M1 = INDEX(B7,T);
    DO WHILE(M1 ¬= 0);
```

```
      IF M2 = 0 THEN CALL LIST(K,M1,M1-D1,M1-D1,Z);
      ELSE CALL LIST(K,M1,M1-D1,M2,Z);
      M1 = NEXT(B7,M1);  END;
      L101: K=K-1;
      RETURN;
     /*****************************************************************/
     /*              SUBROUTINE   NEXT                                */
     /*****************************************************************/
NEXT:            PROC(S,I) FIXED BIN(15);
 /* NEXT RETURNS THE LOCATION OF THE NEXT 1 IN THE BIT
    STRING S AFTER POSITION I  */
   DCL S BIT(24) , SS BIT(24) VAR,
       (I,J) FIXED BIN(15);
     SS=SUBSTR(S,I+1);
     J = INDEX(SS,'1'B);
     IF J = 0 THEN RETURN(J);ELSE RETURN(J + I);
END;
     /*****************************************************************/
     /*                   SUBROUTINE  LIST                            */
     /*****************************************************************/
LIST:            PROC(I,AA,BB,CC,DD);
 /* LIST PLACES THE DECODED MOVE IN THE ARRAY L AT POSITION I */
   DCL (I,AA,BB,CC,DD) FIXED BIN(15);
      IF AA ¬= BB THEN GO TO L99;
      IF CC = DD THEN RETURN;
      L(I,1) = CC; L(I,2) = DD; L(I,3),L(I,4) = 0;
      GO TO XXX99;
  L99: L(I,1)=AA;
      L(I,2)=BB;
      L(I,3)=CC;
      L(I,4)=DD;
  XXX99: IF TRACE THEN PUT SKIP LIST(I||L(I,1)||L(I,2)||L(I,3)||L(I,4));
      L15: K=K+1;
      IF K = 51 THEN DO;
        PUT SKIP LIST('   50 MOVES GENERATED');
        GO TO L101;  END;
      RETURN;
END;
END MVLIST;

     /*****************************************************************/
     /*                                                               */
     /*                    M O D U L E   M V E V A L                  */
     /*                                                               */
     /*****************************************************************/

 MVEVAL: PROC(MVR,MS,LVL,STR,STAT,BD,LST,LSTK,X,KEEPX,C,PE,PH,DFLG,
            REDOX,TRACE,ENT);
```

```
 DCL (MVR,MS,LVL,STR(2,10,3,10),STAT(2,0:12),BD(24),LST(50,4),LISTK)
                          FIXED BIN(15);
 DCL (DEL(0:25),I,J,J1,J2,K,K1,K2,L,L1,L2,P(4),S,TBD(24),PU,PL,T,
     PE(0:6),PH(6,0:5)) FIXED BIN(15),
     (TP,TT,WP,TG,HOLD,TL,KEEPX(3,10), X(50),C(2,10,3,10)) FIXED BIN (31,15),
     EL(0:50) LABEL;
 DCL SAFE(2,6) FIXED BIN(15), (FO,FN,REDOX,TRACE,DFLG,ENT) BIT(1);
 DCL BUFF CHAR(120) VAR, PER CHAR(30) INIT ( (30) '.'),
     TACTIC CHAR(7), TACSTR CHAR(350) INIT CALL SETSTR;
     ON FIXEDOVERFLOW BEGIN; PUT SKIP LIST('OVERFLOW');
     GO TO EX1;  END;
   M = 1;
   IF MVR = 1 THEN MI = 2 ;   ELSE MI = 1;
   IF LVL = 1 THEN X = 0;
       KEEPX(LVL,*) = 0;
  E1:  IF M > LSTK THEN RETURN;
   DEL =0;
        DO I = 1 TO 4; P(I) = LST(M,I);  END;
          DEL(P(1)) = -1;
             DEL(P(2)) = 1;
              IF P(3) = 0 THEN GO TO E2;
                   DEL(P(3)) = DEL(P(3)) - 1;
                   DEL(P(4)) = DEL(P(4)) + 1;
        /*      MAKE TEMPORARY BOARD-TBD      */
   E2: DO I = 1 TO 24;
       TBD(I) = DEL(I) + BD(I);
       IF BD(I) = -1 THEN IF DEL(I) > 0 THEN TBD(I) = TBD(I) + 1;END;
       IF TRACE THEN DO;  PUT SKIP(2) LIST(PER);
       PUT SKIP(2) LIST('MOVE'||M||P(1)||P(2)||P(3)||P(4));
       CALL PUTEV(TBD);  END;
   T = 1;
   E3:  TS = 0;
   L = STR(MVR,MS,LVL,T);
   IF L < 0 | L > 50 THEN DO;
       PUT SKIP LIST('BAD LABEL');
       GO TO EL(0);  END;
   IF ENT THEN IF L-40 < 5 THEN GO TO EX1;
GO TO EL(L);
EX:
   HOLD = C(MVR,MS,LVL,T);
   TT = (TS/64) * HOLD;
   IF TRACE THEN DO;
   TACTIC = SUBSTR(TACSTR,7*L-6,7);
   IF TRACE THEN PUT SKIP LIST('  T='||L||'('||TACTIC||')'||'  TS='||
   TS||'  C='||HOLD||'  TT='||TT); END;
   X(M) = X(M) + TT;
   IF REDOX THEN DO;
       KEEPX(LVL,T) = TT;
```

```
      IF TT >= 0 THEN X(2) = X(2) + TT;
      ELSE X(3) = X(3) + TT;    END;
   EX1: T = T + 1; IF T = 11 THEN GO TO EL(0); ELSE GO TO E3;
EL(0):  IF TRACE THEN PUT SKIP LIST('*** TOTAL MOVE SCORE = '||X(M));
   M = M + 1;  IF M >= 51 THEN DO;
      PUT SKIP LIST('M >= 51 IN MVEVAL');
      M = M - 1; RETURN;  END;
   GO TO E1;
 EL(15):EL(24):EL(25):EL(26):EL(27):EL(28):EL(29):
    GO TO EX;
 EL(1):  /*  OUT MOST MEN   */
         J = STAT(MVR,7);
         S = BD(J);
         EL11: TS = TS - DEL(J)*32;
               IF S = 1 THEN DO;
                    J = STAT(MVR,8);
                    S = 2; GO TO EL11; END;
         GO TO EX;
 EL(2):  /* CLOSE YOUR BOARD   */
         PU = 6;  PL = 1;
               EL21: J1,J2,K1,K2 = 0;
                    /*  COUNT OLD AND NEW HELD POINTS    */
                DO I = PU TO PL BY -1;
                    IF DEL(I) = 0 THEN GO TO EL29;
                        IF BD(I) >= 2 THEN J1 = J1 + 1;
                    IF TBD(I) >= 2 THEN J2 = J2 + 1;
                    IF BD(I) = 1 THEN K1 = K1 + 1;
                    IF TBD(I) = 1 THEN K2 = K2 + 1;
         EL29:END;
               J = J2 - J1;  K = K2 - K1;
               TS = (J + ABS(J))*16 + (K + ABS(K))*8;
 GO TO EX;
 EL(3):  /*  CLOSE HIS BOARD   */
         PU = 24;  PL = 19;
         GO TO EL21;
 EL(4):    /*  DO NOT EXPOSE BLOT   */
       J,K = 0;
       DO I = 1 TO 24;
          IF BD(I) = 1 THEN J = J + 1;
          IF TBD(I) = 1 THEN K = K + 1;
       END;
       TS = (J-K)*16;
         GO TO EX;
 EL(5):   /*  HOLD MAX NO OF POINTS    */
         J,K = 0;
         DO I = 1 TO 24;
               IF BD(I) >= 2 THEN J = J + 2; ELSE IF BD(I) >= 1 THEN J = J + 1;
               IF TBD(I) >= 2  THEN K = K + 2; ELSE IF TBD(I) >= 1 THEN K = K+1;
```

```
        END;
        IF K > J THEN TS = (K-J)*16;
GO TO EX;
EL(6):      /*   BUILD BLOCK     */
        PU = STAT(MVR,11);  PL = STAT(MVR,10);
        J,K,L3,J1,J2,K1,K2,L1,L2 = 0;
        DO I = PU TO PL BY -1;
             J = BD(I);
             IF J < -2 THEN GO TO EL69;
             K = TBD(I);
             IF J = 1 THEN K1 = K1 + 1;
             IF K = 1 THEN K2 = K2 + 1;
        IF J >= 2 THEN DO;
                  J1 = J1 + 1;
                  IF J-2 > 0 THEN L1 = L1 + J - 2;  END;
        IF K >= 2 THEN DO;
                  J2 = J2 + 1;
                  IF K-2 > 0 THEN L2 = L2 + K - 2; END;
        EL69: END;
        J = J2 - J1;
        K = K2 - K1;
        L3 = L2 - L1;
              TS = TS + 32*J + 16*(K + L3);
              IF J*K < 0 THEN TS = TS + 16*J - 8*K;
              IF J*L3 < 0 THEN TS = TS + 16*J - 8*L3;
GO TO EX;
EL(7):    /*  DEPLOY CATCHERS   */
        DO I = 1 TO 4;
             J = DEL(P(I));
             IF BD(I) = -1 THEN K = 0;
                  ELSE K = BD(I);
             IF K + J = 1 THEN TS = TS + 16;
        END;
        GO TO EX;
EL(8):    /*  BLOT IN YOUR INNER TABLE(6-1)   */
        DO I = 1 TO 6;
             J = DEL(I);
             IF J = 0 THEN GO TO EL89;
                  ELSE DO;
                       IF BD(I) = -1 THEN K = 0; ELSE K = BD(I);
                       IF K + J = 1 THEN TS = TS + 16;
                  END;
        EL89: END;
        GO TO EX;
EL(9):    /*  FREEZE RUNNERS   */
        J = STAT(MVR,7);
        IF DEL(J) = 0 THEN TS = 64;
        GO TO EX;
```

```
EL(10):      /*  GET HOME PRONTO    */
    TX = 0;
        DO I = 1 TO 6;
             IF DEL(I) >= 0 THEN TX = TX + DEL(I)*2**(I-1);
             END;
    TS = TX;
        GO TO EX;
EL(11):     /*   BRING HOME 8-PT MEN IN PREF TO 7-PT   */
        IF DEL(8) < 0 THEN IF DEL(7) <= 0 THEN DO;
             IF DEL(7) = -2 THEN GO TO EX;
             ELSE TS = (DEL(7) - DEL(8))*32;  END;
        GO TO EX;
EL(12):     /*    ADVANCE BLOCK   */
        K = STAT(MVR,10) - 1;
        K1 = 2;
    TX = 0;
        EL121:  IF DEL(K) > 0 THEN IF BD(K) > -2 THEN DO;
             IF BD(K) = -1 THEN J1 = 0;
                  ELSE J1 = BD(K);
             IF J1 < 2 THEN DO;
             K2 = J1 + DEL(K);
             IF K2 = 1 THEN TX = TX + 8*K1;
                  ELSE IF K2 >= 2 THEN TX = TX + 16*K1;
             END;
        END;
    IF K1 = 1 THEN DO; TS = TX; GO TO EX; END;
        K1 = 1; K = K - 1;
        GO TO EL121;
EL(13):     /*  MAXIMIZE SAFE MOVES  */
        SAFE = 0;
        PU = STAT(MVR,7);
        FO,FN = '1'B;
        DO I = PU TO 2 BY -1 WHILE(FO | FN);
         IF FO THEN IF BD(I) > 0 THEN DO;
             DO J = 1 TO 6 WHILE(I-J > 0);
                  IF BD(I-J) > -2 THEN IF SAFE(1,J) <2 THEN SAFE(1,J)
                                 = SAFE(1,J) + 1;
                  END; END;
        IF FN THEN IF TBD(I) > 0 THEN DO;
             DO J = 1 TO 6 WHILE (I-J >0);
                  IF TBD(I-J) > -2 THEN IF SAFE(2,J) < 2
                       THEN SAFE(2,J) = SAFE(2,J) + 1; END; END;
              IF FO THEN DO;
                  DO J = 1 TO 6;
                       IF SAFE(1,J) < 2 THEN GO TO EL135;  END;
                   FO = '0'B;  END;
        EL135:  IF FN THEN DO;
             DO J = 1 TO 6;
```

```
                    IF SAFE(2,J) < 2 THEN GO TO EL136; END;
               FN = '0'B;
          EL136:  END;
          END;
          IF FO & FN THEN GO TO EX;
          DO I = 1 TO 6;
               TS = TS + (SAFE(2,I) - SAFE(1,I)) * 2**(I-1);
          END;
          GO TO EX;
EL(14):      /*  LEAVE LONG RUNNER BACK    */
          DO I = 18 TO 13 BY -1;
               IF BD(I) > 0 THEN DO;
                    J = I; K = BD(I);  GO TO EL141;  END;END;
          EL141: IF DEL(J) + K >= 1 THEN TS = 64;
          GO TO EX;
EL(45):      /*   ENTER ON LOW POINTS    */
          IF STAT(MVR,9) = 1 THEN IF  25 - P(2) < 4 THEN TS = 64;
GO TO EX;
EL(16):      /*   MOVE FROM CS TO OUTER TABLE    */
   DO I = 1,3;
               IF P(I) = 13 THEN IF P(I+1) < 6 THEN TS = TS + 32;  END;
GO TO EX;
EL(17):      /*   BEAR OFF MOST MEN POSS   */
          DO I = 2,4;
               IF P(I) = 0 THEN IF P(I-1) > 0 THEN TS=TS+32; END;
GO TO EX;
EL(18):      /*   MAXIMIZE ODDS OF BEARING OFF TWO MEN    */
          /*  FIND LOCATIONS OF TWO LOWEST MEN    */
J1 = 0; K1,K2 = 1;
          DO I = 1 TO 6 WHILE(J1 < 2);
               J2 = TBD(I);
          IF J2 > 0 THEN DO;
               IF J1 = 0 THEN DO; K1=I; J1=1; END;
                    ELSE DO;EL181: K2 = I; J1 = 2; END;
               IF J1 < 2 THEN IF J2 > 1 THEN GO TO EL181;
          END; END;
          TS = 64*(7-K1)*(7-K2)/36;
          GO TO EX;
EL(19):      /*  MAKE HIGH TWO POINTS BOTH EVEN OR ODD  */
          K1,K2 = -1;
          DO I = 6 TO 1 BY -1 WHILE(K2 < 0);
               IF TBD(I) > 0 THEN DO;
                    IF K1 < 0 THEN K1 = I;
                    ELSE K2 = I;  END;
          END;
          IF K1 > 0 THEN IF K2 > 0 THEN DO;
               J1 = TBD(K1);  J2 = TBD(K2);
               K1 = MOD(J1,2);
```

```
            K2 = MOD(J2,2);
        IF MOD(K1+K2,2) = 0 THEN TS = 64;
        END;
GO TO EX;
EL(20):     /*   MOVE TO LOWEST POINTS    */
   GO TO EX;
EL(21):EL(44):     /*   COVER BLOTS    */
     J,K = 0;
        DO I = 1 TO 4;
        IF BD(I) = 1 THEN J = J + 1;
        IF TBD(I) = 1 THEN K = K + 1;  END;
     IF J > K THEN TS = (J-K)*16;
GO TO EX;
EL(22):     /*  DISTRIBUTE MEN IN OUTER TABLES    */
        J1,J2 = 0;
        DO I = 7 TO 18;
            IF BD(I) > 0 THEN J1 = J1 + 1;
            IF TBD(I) > 0 THEN J2 = J1 + 1;
        END;
        TS = (J2 - J1)*16;
        GO TO EX;
EL(23):    /*  BRING MEN INTO SAVE AREA BEHIND OPPONENT'S
                                    REAR-MOST MEN   */
        J = STAT(MI,7);
        DO I = 2,4;
            IF P(I) > J THEN IF P(I+1)   < J THEN  TS = TS + 32;   END;
        GO TO EX;
EL(46):EL(47):EL(48):EL(49):
/*      ENTER AND ...          */
        J1 = 1;
            EL292: IF P(J1) ¬= 25 THEN GO TO EL293;
            J2 = BD(P(J1+1));
        /*   HIT    */
        IF L = 46 THEN IF J2 = -1 THEN GO TO EL294;
        /*  DO NOT HIT    */
        IF L = 47 THEN IF J2 ¬= -1 THEN GO TO EL294;
        /*  COVER   */
        IF L = 48 THEN IF J2 = 1 THEN GO TO EL294;
    /*  BLOT  */
        IF L = 49 THEN IF J2 = -1 | J2 = 0 THEN GO TO EL294;
       GO TO EL293;
        EL294: TS = TS + 32;
  EL293: IF J1 = 3 THEN GO TO EX ;
         J1 = 3;
    GO TO EL292;
EL(30): EL(31):EL(32):EL(33):EL(34):EL(35):EL(36):EL(37):EL(38):EL(39):
    /*    OCCUPY KEY POINTS    */
IF L < 33 THEN DO;
```

```
  IF L = 30 THEN J1 = 20;
        ELSE IF L = 31 THEN J = 21;
        ELSE IF L = 32 THEN J1 = 18;
  END;
 ELSE J1 = L - 30;
  J2 = DEL(J1);
  IF BD(J1) > 1 THEN IF J2 > 0 THEN GO TO EX;
  IF J2 < 0 THEN IF TBD(J1) > 1 THEN GO TO EX;
  TS = DEL(J1) * 32;
        GO TO EX;
EL(40):EL(41):EL(42):EL(43):
        /*   HIT BLOTS(ALL,OUTER TABLES, HIS INNER, YOURINNER */
  DO I = 2,4;
        J = P(I);
        K = BD(J);
        IF J ¬= 0 THEN IF K = -1 THEN DO;
              IF L = 40 THEN GO TO EL432;
              ELSE IF L = 41 THEN IF J < 19 THEN IF J > 6
                   THEN GO TO EL432;
              ELSE IF L = 42 THEN IF J > 18 THEN GO TO EL432;
              ELSE IF L = 43 THEN IF J < 7 THEN GO TO EL432;
              GO TO EL433;
        EL432:  TS =  TS + 32;
              EL433:  END;
        END;
GO TO EX;
EL(50):     /*   COMPUTE RISK FACTOR   */
        IF DFLG = '0'B THEN GO TO EX;
  K3,TG,TL = 0;
  DO I1 = 1 TO 24;
     K = 0;
     IF TBD(I1) ¬= 1 THEN GO TO EL509;  /* NOT A BLOT */
     K3 = K3 + 1;  /* TALLY UP NO. OF BLOTS  */
 /* G A I N  */
     IF BD(I1) = -1 THEN DO;  /*  HIS BLOT HAS BEEN HIT  */
J3 = STAT(MVR,5);
IF J3 < 0 | J3 > 6 THEN DO;
    PUT SKIP LIST('STAT(MVR,5) IS BAD');
    GO TO EX; END;
        WP = PE(J3);
        IF WP = 0 THEN WP = .1;
        TG = TG + I1 + ((1/WP)-1)*8.17;
        K = 1; END;
 /* L O S S  */
  K2, K1, TP = 0;
     IF STAT(MVR,5) = 6 THEN GO TO EL509;
     DO I2 = 1 TO 6;
        I3 = I1 - I2;
```

```
          IF I3 < 1 THEN GO TO EL502; /* OFF THE BOARD  */
          J1 = TBD(I3);  /* SCAN DOWN THE BOARD   */
          IF J1 > 1 THEN K1 = K1 + 1;
          ELSE IF J1 < 0 THEN DO;
             TP = TP + PH(I2,K1);
             K2 = K2 + 1;  END;
          IF I1 < 7 THEN DO;
             K4 = STAT(MI,9) + K;
         IF K4 < 0 THEN DO;
           PUT SKIP LIST('BAD STAT(MI,9)' );
               GO TO EX; END;
             IF K4 > 2 THEN K4 = 2;
             TP = TP + (K4 * PE(STAT(MVR,6))/6);
             K2 = K2 + 1;
          END;
       END;
       EL502:  IF K2 ¬= 0 THEN TP = TP/K2;
 IF TP > 1 THEN TP = 1;
               J3 = STAT(MI,6);
                 IF J3 < 0 | J3 > 6 THEN DO;
                   PUT SKIP LIST('BAD STAT(MI,6)');
                   GO TO EX;  END;
           WP = PE(STAT(MI,6));
          IF WP = 0 THEN WP = .1;
 TX = 25 - I1 + (8.17 * ((1/WP)-1));
 TL = TL + (TP * TX);
    EL509: ;END;
          IF K3 ¬= 0 THEN TS = 16 * ((TG - TL)/K3);
          GO TO EX;
SETSTR: PROC;
TACSTR = 'REARMENCLOSEYRCLOSEHSNOXPOSEMAXPTS BLDBLCKCATCHERBLOTYTB'||
'FRZRNRSGETHOME8BEFOR7ADVBLOKMAXSAFELONGRUNENTLOW CSTOOUTBOMOST MXBOODSHI2EVEN'
||'TOLOWPTCOVER  DISTRIBSAVAREAUNUSED UNUSED UNUSED UNUSED UNUSED UNUSED '||
'HIS 5  HIS 4  HIS 7  YOUR 3 YOUR 4 YOUR 5 YOUR 6 YOUR 7 YOUR 8 YOUR 9 '||
'HITALL HITOUTRHITHSINHITYRINUNUSED ENTERLOENTRHITENTNOHTENTOCVR'||
'ENTNOCV RISK  ';  RETURN;
 END MVEVAL;

     /*****************************************************************/
     /*                                                               */
     /*                M O D U L E     M V P I C K                    */
     /*                                                               */
     /*****************************************************************/

MVPICK: PROC(L,LCT,S,T,DEL,TRACE);
   /* MVPICK FINDS THE MOVE WHICH HAS THE MAX SCORE AND PLACES IT AND
      THE CORRESPONDING MOVE AT THE TOP OF THE LIST.
      IF MORE THAN ONE MOVE HAS THE SAME MAX
```

```
      SCORE (SMAX) THEN MVPICK REORDERS THE LIST OF MOVES (L) PUTTING
      THESE MOVES AT THE TOP OF THE LIST AND RETURNING THE NUMBER OF
      SUCH MOVES (LCT) ALONG WITH A BIT CODE (T) INDICATING THAT A TIE
      EXISTS    */
   DCL (L(50,4), LCT,IMAX,K,TEMPL(4)) FIXED BIN(15),
   (TRACE,T) BIT(1), (S(50),SMAX,DEL,TEMPS) FIXED BIN(31,15);
   IMAX=1;
   SMAX=S(1);
   DO I=2 TO LCT;
   IF S(I) > SMAX THEN DO;
   SMAX = S(I);
   IMAX = I; END;
   END;
   IF IMAX = 1 THEN GO TO SORT;
   TEMPL(*) = L(IMAX,*);
   TEMPS = S(IMAX);
   L(IMAX,*) = L(1,*);
   S(IMAX) = S(1);
   S(1) = TEMPS;
   L(1,*) = TEMPL(*);
   IMAX = 1;
   SMAX = S(1);
   SORT: K = 2;
   DO I = 2 TO LCT;
     IF SMAX - S(I) <= DEL THEN DO; IF K¬=I THEN DO;
      L(K,*) = L(I,*);
       S(K) = S(I);END;
       K = K + 1;  END;
   END;
   IF K = 2 THEN T = '0'B;
     ELSE T = '1'B;
   LCT = K-1;
   IF TRACE THEN DO;
      DO I = 1 TO LCT;
   PUT SKIP LIST('    '||I||L(I,1)||L(I,2)||L(I,3)||
  L(I,4)||S(I));END;END;
         RETURN;
END MVPICK;

     /*****************************************************************/
     /*                                                               */
     /*                    M O D U L E   M V I M P                    */
     /*                                                               */
     /*****************************************************************/

 MVIMP:  PROC(M,D1,D2,S,WB,GB,MV,SCR,BLOT,BO,T,F,TRACE);
         /*  MVIMP ACCEPTS THE SELECTED MOVE AND IMPLEMENTS IT, I.E.,
         UPDATES THE GAME BOARD AND APPROPRIATE GAME STATISTICS    */
```

```
DCL      (M,  /*  MOVER, 1 = WH, 2 = BL    */
          D1,D2,   /*  THE DICE ROLL     */
          S(2,0:12),  /* STATISTICS MATRIX    */
         ((WB,GB,TB) (24)),     /* WORK,GAME, TEMP BOARDS  */
         MV(4),  /* THE CURRENT MOVE    */
         I,I1,I2,J,J1,J2,K,K1,K2,MI,P,BH,BT)    /*WORKING VARIABLES */
                   FIXED BIN (15),
         (BO, /*  BEAR OFF FLAG, 1 = BEARING OFF   */
         TRACE,SF,BLOT(2,24)) BIT(1),     /* LOCATION OF BLOTS   */
         ((T,F)(2)) BIT(24),       /*  LEGAL TO/FROM MOVE POSITIONS  */
         SCR FIXED BIN(31,15),   /*  SCORE OF CURRENT MOVE    */
         BUFF CHAR(120) VAR, YN CHAR(1);
   IF M = 1 THEN MI = 2; ELSE MI = 1;
   IF D1*D2 ¬= 0 THEN DO;
      SF = '0'B;  GO TO II1; END;
      ELSE DO;  SF = '1'B;
         PUT SKIP LIST('    BEAROFF? 1 OR 0') ; GET LIST(J) ;
         IF J = 1 THEN BO = '1'B;
         ELSE BO = '0'B;
      IF M= 1 THEN DO; WB,TB = GB; END;
         ELSE DO;DO I = 1 TO 24; WB(I) = -GB(25-I); END; TB=WB;END;
         GO TO I3;
   END;
   II1:  /*   MAKE TEMP BOARD(TB) = WORKBOARD + MOVES    */
   TB = WB;
   DO J = 1,3;
      IF MV(J) = MV(J + 1) THEN GO TO II2;
         J1 = MV(J);
      IF J1 ¬= 25 THEN DO;
         IF TB(J1)  <= 0 THEN DO;
         IF J = 3 THEN IF MV(2) = MV(3) THEN GO TO II11;
            PUT SKIP LIST('ATTEMPTING TO IMPLEMENT'||
      ' AN ILLEGAL MOVE.  IT WILL BE IGNORED.');
            GO TO II2; END;
        II11: IF J1 ¬= 25 THEN TB(J1) = TB(J1) - 1; END;
        J2 = MV(J+1);
      IF J2 ¬= 0 THEN DO;
         IF TB(J2) = -1 THEN TB(J2) = 0;
         TB(J2) = TB(J2) + 1; END;
   II2: ; END;
         /*   UPDATE GAMEBOARD    */
  IF M = 1 /* WHITE  */ THEN GB = TB;
   ELSE  /* BLACK   */  DO;
       DO I = 1 TO 24;
              GB(I) = -TB(25-I); END;  END;
I3:  /* UPDATE NO: OF MEN LEFT TO BEAR */
  IF SF THEN DO;
      PUT SKIP LIST('   ENTER NO. OF MEN LEFT TO BEAR: W,B');
```

```
        GET  LIST((S(I,3) DO I = 1,2));
     GO TO I7; END;
     K=0;
   DO I = 2,4;
        IF MV(I)=0 THEN IF MV(I-1) ¬= 0 THEN K = K + 1;END;
    S(M,3) = S(M,3) - K;
 XXX2:  IF TRACE THEN PUT SKIP LIST('MEN TO BEAR:'||S(M,3));
I7: I8:/* UPDATE LOCATION OF REARMOST / NEXT TO REARMOST MEN */
    K1,K2=0;
    DO I=24 TO 1 BY-1 WHILE(K1=0);
        IF GB(I)>0 THEN K1=I; END;
    DO I=K1-1 TO 1 BY -1 WHILE(K2=0);
        IF GB(I)>0 THEN K2=I; END;
 XXX3: IF TRACE THEN PUT SKIP LIST('WHITE REAR MEN:'||K1||K2);
    S(1,7)=K1;
    S(1,8)=K2;
    K1,K2 = 0;
    DO I = 1 TO 24 WHILE(K1=0);
        IF GB(I) < 0 THEN K1 = I;  END;
    DO I = K1 + 1 TO 24 WHILE(K2 = 0);
        IF GB(I) < 0 THEN K2 = I;  END;
    IF TRACE THEN PUT SKIP LIST('BLACK REAR MEN:'||K1||K2);
    S(2,7) = 25 - K1;
    S(2,8) = 25 - K2;
I9:    /* NO. OF OPPONENT'S MEN ON THE BAR */
   IF SF THEN DO;
      PUT SKIP LIST('   ENTERNO. OF MEN ON THE BAR: W,B') ;
        GET  LIST ((S(I,9) DO I = 1,2));
        GO TO I13; END;
    J1 = 0;
    DO I=2,4;
        J2= MV(I);
        IF J2 ¬=0 THEN IF J2 ¬= 25 THEN IF WB(J2)=-1 THEN J1=J1+1; END;
    IF J1 = 2 THEN IF MV(2) = MV(4) THEN J1 = 1;
I99:   S(MI,9) = S(MI,9) + J1;
 XXX4: IF TRACE THEN PUT SKIP LIST('PASSIVE MEN ON BAR:'||S(MI,9));
 /*  ACTIVE MEN ON BAR     */
    J1 = S(M,9);
    DO I = 1,3;
    IF MV(I) = 25 THEN IF MV(I+1) ¬= 25 THEN J1 = J1-1;END;
    IF TRACE THEN PUT SKIP LIST('ACTIVE MEN ON BAR:'||J1);
    S(M,9) = J1;
I13:  /* UPDATE BLOT ARRAY */
    BLOT='0'B;
    DO I=1 TO 24;
        J=TB(I);
        IF J=1 THEN BLOT(M,I)='1'B;
        ELSE IF J=-1 THEN BLOT (MI,25-I)='1'B;
```

```
   END;
 XXX5: IF TRACE THEN PUT SKIP LIST('BLOT: WHT||BLK:',STRING(BLOT));
   IF SF THEN GO TO I01;
I0:    /* UPDATE CUMULATIVE DICE COUNT */
   S(M,0)=S(M,0)+D1+D2;
 XXX6: IF TRACE THEN PUT SKIP LIST('CUM.DICE:'||S(M,0));
I01:    /* UPDATE ADVANCE */
   L1,L2,J1,J2 = 0;
      DO I=1 TO 24;
         K=TB(I);
          IF K > 0 THEN DO;
             J1 = J1 + K*(25-I);
             IF I > 6 THEN L1 = L1 + K*(I-6);END;
          IF K < 0 THEN DO;
             J2 = J2 - (K*I);
             IF I < 19 THEN L2 = L2 - (K*(19-I));END; END;
      /* ADJUST ADVANCE FOR EACH MAN ALREADY BORNE OFF (+25)  */
   J1=J1+ (15-S(M,3))*25;
   J2 = J2 + (15-S(MI,3))*25;
   S(M,1)= J1 - 208;
   S(MI,1) =  J2 - 208;
   IF SF THEN S(*,0) = S(*,3);
 XXX7: IF TRACE THEN PUT SKIP LIST('ADVANCE:WHT,BLK:'||S(1,1)||S(2,1));
I02: /* UPDATE BEAROFF DISTANCE   */
   S(M,2) = L1 + (S(M,9)*20);
   S(MI,2) = L2 + (S(MI,9)*20);
 XXX8: IF TRACE THEN PUT SKIP LIST('BEAROFF DIST:WHT,BLK'||S(1,2)||S(2,2));
I4:    /* UPDATE OVERLAP */
   J1=S(M,7);/* LOCATION OF YOUR REARMOST MEN */
   J2=25-S(MI,7);/* LOCATION OF HIS REARMOST MEN */
      /* OVERLAP = PT DISTANCE SEPARATING HIS REARMOST MEN FROM YOURS
      TIMES THE SUM OF THE NUMBER OF MEN ON THESE POINTS. */
   J = J1 - J2 + 1;
   IF J<0 THEN K=0; /* NO OVERLAP */
   ELSE K=J*(TB(J1)-TB(J2));
 XXX9: IF TRACE THEN PUT SKIP LIST('OVERLAP:'||K);
   S(1,4)=K;
I5:I6:/* UPDATE NO OF YOUR/OPPONENTS HOME POINTS HELD */
  J1 , J2 = 0;
  DO I = 1 TO 6;
     J = TB(I);
     IF J > 1 THEN J1 = J1 + 1;
     IF J < -1 THEN J2 = J2 + 1;
  END;
  S(M,5) = J1;
  S(MI,6) = J2;
  K1, K2 = 0;
  DO I = 19 TO 24;
```

```
      K = TB(I);
      IF K > 1 THEN K1 = K1 + 1;
      IF K < -1 THEN K2 = K2 + 1;
   END;
   S(M,6) = K1;
   S(MI,5) = K2;
   IF TRACE THEN DO;
      PUT SKIP LIST('WHITE HOME PTS HELD BY WHITE:'||S(1,5));
      PUT SKIP LIST('WHITE HOME PTS HELD BY BLACK:'||S(2,6));
      PUT SKIP LIST('BLACK HOME PTS HELD BY BLACK:'||S(2,5));
      PUT SKIP LIST('BLACK HOME PTS HELD BY WHITE:'||S(1,6));
   END;
I10:I11:/*UPDATE LOCATION OF HEAD/TAIL OF BLOCK */
    IF SF THEN DO;
       PUT SKIP LIST('    ENTER HEAD OF BLOCKS, W,B');
       GET LIST(J1,J2);
       S(1,10) = J1;
       S(1,11) = J1 + 5;
       S(2,10) = 25 - J2;
       S(2,11) = 30 - J2;
       GO TO I12;  END;
    J1=S(M,10); /* PREVIOUS HEAD OF BLOCK */
    IF J1 = 5 THEN J1 = 6;
I101: K1=J1;
    DO J=J1-1 TO J1-2 BY -1 WHILE(J>0);
       IF TB(J)>1 THEN K1=J; END;
 XXX11: IF TRACE THEN PUT SKIP LIST('ACTIVE HD/TAIL OF BLOCK:'||K1||K1+5);
    S(M,10)=K1;
    S(M,11)=K1+5;
I12: /* UPDATE BLOCK POTENTIAL,BP/A MEASURE OF HOW NEAR WE ARE TO
       COMPLETING A PERFECT BLOCK. A PERFECT SCORE, 600, INDICATES A
       PERFECT BLOCK EXISTS. BP IS AWARDED 100 PTS FOR EACH PT HELD BY
       2 OR MORE MEN. ADDITIONAL POINTS (UP TO 99) ARE AWARDED ON THE
       BASIS OF THE POTENTIAL FOR IMPROVING THE BLOCK. THE TECHNIQUE
       IS TO FIND A GAP IN THE BLOCK AND ESTIMATE THE PROBABILITY OF
       FILLING THE GAP */
    K,K1,K2,J=0;
    BH=S(M,10);
    BT=S(M,11);
    DO I1=BH TO BT  WHILE(I1<25);
       J1=TB(I1);
       IF J1>1 THEN K1 = K1 + 1000;
       IF J1 < -1 THEN K1 = K1 + 100;
          ELSE DO; J=J+1; /* GAP COUNT */
             IF J1=1 THEN P=2; ELSE P=1;
          DO I2=I1+1 TO I1+6  WHILE(I2<25);
             J2=TB(I2);
             IF I2>BT THEN DO; IF J2>0 THEN K2=K2+P; END;
```

```
            ELSE IF J2>2 THEN K2=K2+P;
          END;END;
       ENDL:;END;
  IF J=0 THEN S(M,12)=K1;
  ELSE S(M,12)=K1+ FLOOR(((K2/J)/12)*100)-1;
 XXX12: IF TRACE THEN PUT SKIP LIST('ACTIVE BLOCK POTENTIAL:'||S(M,12));
 I15:  /* UPDATE TO/FROM STRINGS    */
  T(1),T(2) = (24) '1'B; F(1),F(2) = (24) '0'B;
  DO I = 1 TO 24;
       J = GB(I);
       IF J > 0 THEN DO;
         SUBSTR(F(1),25-I,1) = '1'B;
         IF J > 1 THEN SUBSTR(T(2),I,1) = '0'B; END;
       IF J < 0 THEN DO;
         SUBSTR(F(2),I,1) = '1'B;
            IF J < -1 THEN SUBSTR(T(1),25-I,1) = '0'B; END;
    END;
 XXX13: IF TRACE THEN DO; PUT SKIP LIST('WHT TO_STR:'||T(1));
   PUT SKIP LIST('WHT FM_STR:'||F(1));
   PUT SKIP LIST('BLK TO_STR:'||T(2));
   PUT SKIP LIST('BLK FM_STR:'||F(2)); END;
   I50: RETURN;
END MVIMP;

     /*****************************************************************/
     /*                                                               */
     /*                    M O D U L E   P U T 1                      */
     /*                                                               */
     /*****************************************************************/

 /* PUT1 HANDLES THE OUTPUT OF THE PROGRAM. THE OUTPUT MAY BE
    DIRECTED TO A FILE(GAMOUT) IF DESIRED. AN ABBREVIATED FORM
    OF THE BOARD IS DISPLAYED AT THE TERMINAL. */
       PUT1:   PROC(N,M,GB,MV,CS,STAT,D1,D2,SC,BATCH,GAMFLAG)  ;
DCL (N,M,((TB,XB,GB)(24)),MV(2,4),STAT(2,0:12),CS(2),D1,D2,J,K
                    )FIXED BIN (15);
DCL (C1,C2) CHAR(8),
   GAMOUT FILE STREAM ENVIRONMENT(F(80)),
   KEEPCS(2) FIXED BIN(15) STATIC,
   SC FIXED BIN (31,15),
   (BATCH,EFLAG,GAMFLAG) BIT(1),
   C CHAR (2),
   TEMP CHAR(54)
         INITIAL('   | . . . . . | . . . . . | . . . . . | . . . . . |'),
   BD CHAR(51) INIT('** 24        19 18        13 12        7 6        1'),
   STAR20 CHAR(20) INITIAL ((20) '*'),
   STAR2 CHAR(2) INITIAL ('**'),
   SLAB CHAR(56) INITIAL
```

```
   ('POSITIONRUN-OFF RUN-DEF BLOCKINGBACK    BO-OFF  BO-DEF '),
   MVR CHAR(5),
   BUFF(5) CHAR(80) VAR,
   DIG PICTURE'Z',
   SCRPIC PICTURE 'ZZZ9.999',
   H CHAR(1) INITIAL ('-'), CM CHAR(3) INITIAL (' , '),
   (P1,P2,P3,P4,P5,P6,P7,P8,P9,P10) PICTURE'Z9',
   PM PICTURE'ZZ9',(PD1,PD2) PICTURE'9';
   EFLAG = '0'B;
   BUFF = '';
   IF N = 0 THEN DO;
      BUFF(1) = '**   INITIAL BOARD SETUP';
      KEEPCS = CS;
      GO TO XP0; END;
   IF M = 1 THEN MVR = 'WHITE'; ELSE DO;
      IF D1 = D2 THEN K = 2; ELSE K = 1;
  DO I = 1 TO K; DO J = 1 TO 4;
      MV(I,J) = 25 - MV(I,J);  END; END;
         MVR = 'BLACK';
         END;
   PM = N;  PD1 = D1;  PD2 = D2;
   IF MV(1,1) = MV(1,2) THEN DO;
   BUFF(1) = '**   MOVE NO: '||PM||'    ROLL: '||PD1||H||PD2||
   '     '||MVR||'    NO MOVES';
   IF ¬BATCH THEN DO;PUT SKIP(3) LIST(BUFF(1)); RETURN;END;
   ELSE GO TO XP0 ;END;
   P1 = MV(1,1);P2=MV(1,2);P3=MV(1,3);P4=MV(1,4);
   BUFF(1) = '**   MOVE NO: '||PM||'    ROLL: '||PD1||H||PD2||
   '     '||MVR||'    '||P1||H||P2;
   IF P3 ¬= P4 THEN BUFF(1) = BUFF(1)||CM||P3||H||P4;
   IF D1 = D2 THEN DO;
   P5=MV(2,1);     P6=MV(2,2);P7=MV(2,3);P8=MV(2,4);
   IF P5 ¬= P6 THEN BUFF(1) = BUFF(1)||CM||P5||H||P6;
   IF P7 ¬= P8 THEN BUFF(1) = BUFF(1)||CM||P7||H||P8;
  END;
XP0: XB = GB;
   GO TO XP1;
   PUTEV: ENTRY(TB);
   BUFF = '';
   XB = TB;
   EFLAG = '1'B;
XP1: BUFF(2) = STAR2||'  ';
   DO I = 24 TO 1 BY -1;
      J = ABS(XB(I));
      IF J > 2 THEN DIG = J - 2;
        ELSE DIG = 0;
      BUFF(2) = BUFF(2)||DIG||' ';
   END;
```

```
XP2: BUFF(3) = '**  ';
   DO I = 24 TO 1 BY -1;
      J = XB(I);
      IF J > 1 THEN C = 'O ';
      ELSE IF J < -1 THEN C = 'X ';
      ELSE C = '  ';
      BUFF(3) = BUFF(3)||C;   END;
   IF EFLAG THEN GO TO XP3;
   J = STAT(1,9);  K = STAT(2,9);
   IF J = 0 THEN IF K = 0 THEN GO TO XP3;
   P9 = J;  P10 = K;
   BUFF(3) = BUFF(3)||'  BAR: W: '||P9||'  B:'||P10;
XP3: BUFF(4) = '**  ';
   DO I = 24 TO 1 BY -1;
      J = XB(I);
      IF J > 0 THEN C = 'O ';
      ELSE IF J < 0 THEN C = 'X ';
      ELSE C = '  ';
   BUFF(4) = BUFF(4)||C; END;
   IF SC = 0 THEN GO TO XP4;
   BUFF(4) = BUFF(4)||'  MOVE SCORE:'||SC;
 XP4:DO I = 4 TO 54 BY 2;
        SUBSTR(BUFF(4),I,1) = SUBSTR(TEMP,I,1);  END;
XP5: IF EFLAG THEN GO TO PRNT;
   IF BATCH = '0'B THEN IF CS(M) = KEEPCS(M) THEN GO TO PRNT;
   KEEPCS(M) = CS(M);
   J = CS(1);
   K = CS(2);
   C1 = SUBSTR(SLAB, 8*J-7,8);
   C2 = SUBSTR(SLAB,8*K-7,8);
   BUFF(5) = '**   STRATEGIES:  W: '||C1||'  B: '||C2;
   PRNT: IF EFLAG THEN DO;
   PUT SKIP(3) LIST(BUFF(2));
   DO I = 3,4;
   PUT SKIP LIST(BUFF(I)); END;
   PUT SKIP LIST(BD);
RETURN; END;
IF ¬BATCH THEN DO;
   PUT SKIP(3) LIST(BUFF(1));
   K = 2;
   IF LENGTH(BUFF(2)) > 4 THEN DO;
      PUT SKIP(2) LIST(BUFF(2));
      K = 1;  END;
   IF LENGTH(BUFF(3)) > 4 THEN DO;
      PUT SKIP(K) LIST (BUFF(3));
      K = 1;  END;
   PUT SKIP(K) LIST(BUFF(4));
   PUT SKIP LIST(BD);
```

```
    IF LENGTH(BUFF(5)) ¬= 0 THEN PUT SKIP(2) LIST(BUFF(5));
    RETURN; END;
  IF ¬GAMFLAG THEN DO;
    PUT SKIP(4) LIST (STAR20);
    PUT SKIP LIST(STAR2);
    PUT SKIP LIST(BUFF(1));
    PUT SKIP LIST(STAR2);
    DO I = 2 TO 4; PUT SKIP LIST(BUFF(I)); END;
    PUT SKIP LIST(BD);
    PUT SKIP LIST(STAR2);
    PUT SKIP LIST(BUFF(5));
    PUT SKIP LIST(STAR2);
    PUT SKIP LIST(STAR20);
    RETURN; END;
IF GAMFLAG THEN DO;
    OPEN FILE(GAMOUT) OUTPUT;
    PUT FILE(GAMOUT) EDIT(STAR20)(SKIP(4),COL(1),A(20));
    PUT FILE(GAMOUT) EDIT(STAR2)(SKIP(1),COL(1),A(2));
    PUT FILE(GAMOUT) EDIT(BUFF(1))(SKIP(1),COL(1),A(80));
    PUT FILE(GAMOUT) EDIT(STAR2)(SKIP(1),COL(1),A(2));
    DO I = 2 TO 4; PUT FILE(GAMOUT) EDIT(BUFF(I))(SKIP(1),COL(1),A(80)); END;
    PUT FILE(GAMOUT) EDIT(BD)(SKIP(1),COL(1),A(51));
    PUT FILE(GAMOUT) EDIT(STAR2)(SKIP(1),COL(1),A(2));
    PUT FILE(GAMOUT) EDIT(BUFF(5))(SKIP(1),COL(1),A(80));
    PUT FILE(GAMOUT) EDIT(STAR2)(SKIP(1),COL(1),A(2));
    PUT FILE(GAMOUT) EDIT(STAR20)(SKIP(1),COL(1),A(20));
    CLOSE FILE(GAMOUT);
    RETURN; END;
END PUT1;
```

LIST OF REFERENCES

1. Feigenbaum, E. A. and Feldman, J., Ed., Computers and Thought, p. 37, McGraw-Hill, 1963.

2. Samuel, A. L., Some Studies in Machine Learning Using the Game of Checkers, IBM Journal, p. 211, July 1959.

3. Newell, A., Shaw, J. C., and Simon, H. A., Chess-Playing Programs and the Problem of Complexity, in Computers and Thought, p. 39.

4. Newman, C., and Uhr, L., BOGART: A Discovery and Induction Program for Games, Proceedings, ACM 20th National Conference, p. 176, 1965.

5. Samuel, A. L., Some Studies in Machine Learning Using the Game of Checkers. II -- Recent Progress, IBM Journal, p. 601, November 1967.

6. Mabardi, G., Backgammon to Win, Horace Liveright, 1930.

7. Richard, W. L., Complete Backgammon, David McKay Company, Inc., 1940.

8. Backgammon and Draughts, Frederick A. Stokes Co., publication date unknown.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	LTJG Robert Bolles, USNR Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	LCDR William E. Bushey, USN 84 Carlton Drive Monterey, California 93940	1

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

Gammon, an Approach to the Concept of Strategy in Game-Playing Programs

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Master's Thesis; December 1970

5. AUTHOR(S) (First name, middle initial, last name)

William Edward Bushey

6. REPORT DATE December 1970	7a. TOTAL NO. OF PAGES 141	7b. NO. OF REFS 8

8a. CONTRACT OR GRANT NO. b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

Gammon is a computer program which plays Backgammon. It was developed to investigate a concept of strategy as an integrating and driving force in the play of the game. A strategy is defined and implemented as a set of tactics where each tactic is a specific feature of play. Moves are generated and evaluated on the basis of the degree to which they accomplish the objectives of the tactics making up the current strategy. Strategies are selected and changed during the course of a game by heuristic analysis of the current game situation. Deductive learning mechanisms are employed to improve the program's performance, against both human and virtual machine opponents.

DD FORM 1 NOV 65 1473 (PAGE 1)
S/N 0101-807-6811

UNCLASSIFIED
Security Classification

A-31406

14. KEY WORDS	LINK A ROLE	LINK A WT	LINK B ROLE	LINK B WT	LINK C ROLE	LINK C WT
Computer Games						
Game strategies						
Backgammon						
Machine learning						

Gaylord
SHELF BINDER
Syracuse, N. Y.
Stockton, Calif.

2 NOV 72 21020
29 OCT 73 22343
17 NOV 77 22343
14 NOV 74
14 NOV 74 22651
14 NOV 74
12 DEC 79 26005

Thesis 124419
B924 Bushey
c.1 Gammon, an approach to the concept of strategy in game-playing programs.
2 NOV 72 21020
29 OCT 73 22343
17 NOV 77 22343
14 NOV 74
14 NOV 74 22651
12 DEC 79 26005

Thesis 124419
B924 Bushey
c.1 Gammon, an approach to the concept of strategy in game-playing programs.

thesB924
Gammon :
3 2768 002 08835 3
DUDLEY KNOX LIBRARY